FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Centralizované plánování autonomní dopravy
Student:	Bc. Ondřej Pleticha
Vedoucí:	doc. RNDr. Pavel Surynek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Předpokládejme scénář budoucího vývoje dopravy, která je zcela autonomní a řízená centrálně. Uživatelé přicházejí s požadavky odkud, kam a kdy chtějí jet a systém jim zařídí, že je autonomní vozidlo v požadovaném místě a čase vyzvedne a dopraví do cíle. Cílem práce je navrhnout a otestovat algoritmy pro řízení takového systému. Klíčovou otázkou přitom je plánování a rozvrhování jízd jednotlivých autonomních vozidel. Lze rovněž vzít v úvahu různé účelové funkce, jako je plynulost dopravy či celková energetická náročnost systému. Předpokládáme, že uchazeč bude postupovat podle následující osnovy:

1. Seznamte se s problémem centralizovaného řízení dopravy ve městech.
2. Navrhněte systém pro řízení vozidel s diskrétními kroky zejména plánovací a rozvrhovací modul.
3. Navrhněte a implementujte simulaci pro testování navržených algoritmů.
4. Pomocí simulací algoritmy vyhodnoťte.

Seznam odborné literatury

[1] Hang Ma, Wolfgang Hönig, T. K. Satish Kumar, Nora Ayanian, Sven Koenig: Lifelong Path Planning with Kinematic Constraints for Multi-Agent Pickup and Delivery. AAAI 2019: 7651-7658

[2] Lukáš Chrpa, Daniele Magazzeni, Keith McCabe, Thomas Leo McCluskey, Mauro Vallati. Automated planning for Urban traffic control: Strategic vehicle routing to respect air quality limitations. Intelligenza Artificiale 10(2): 113-128 (2016)

[3] Stuart Russell, Peter Norvig: Artificial Intelligence: A Modern Approach (3rd edition), Pearsons, 2009.

Ing. Karel Klouda, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 8. února 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Centralizované plánování autonomní dopravy

Bc. Ondřej Pleticha

Katedra aplikované matematiky
Vedoucí práce: doc. RNDr. Pavel Surynek, Ph.D.

27. května 2020

Poděkování

Chtěl bych touto cestou poděkovat vedoucímu této práce **doc. RNDr. Pavlu Surynkovi Ph.D.** za jeho cenné odborné rady, konzultace a připomínky.

Neméně chci také poděkovat své přítelkyni **Bc. Zuzaně Hánové** za její rady, podporu a cit pro český jazyk.

Na závěr bych chtěl poděkovat svým rodičům za jejich podporu při celém studiu.

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.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 27. května 2020 .....

Odkaz na tuto práci

Pleticha, Ondřej. *Centralizované plánování autonomní dopravy.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato diplomová práce se zabývá problémem řízení autonomní dopravy ve městech. Je uvažována budoucnost, kde jsou všechna vozidla sdílená a dokáží přepravovat osoby či předměty sama bez řidiče. Zákazníci pouze vytváří požadavky o přepravu. Cílem práce je pro daný systém navrhnout a otestovat algoritmy, které by mohly být použity při řešení problému nalezení vhodných tras pro vozidla. Klíčové ukazatele jsou celková energetická náročnost systému a spokojenost zákazníků. Pro ověření výsledků algoritmů je vytvořena simulace, která ve zjednodušené verzi napodobuje dopravu a požadavky ve městě. Nejprve je definováno prostředí, které popisuje, jak by uvedený dopravní systém mohl fungovat. Pro řízení dopravy jsou uvažovány tři různé existující algoritmy, které se liší v rozdělování požadavků a plánování tras vozidel. Všechny tři algoritmy jsou upraveny a vylepšeny o možnost zpracovávat požadavky s různou prioritou. Testování je zaměřeno na schopnost algoritmů uspokojovat požadavky při různém poměru vozidel a úkolů. Dále je také zkoumáno, jak dlouho zákazníci za určitých podmínek čekají nebo jaké jsou požadavky na výpočetní výkon u jednotlivých algoritmů.

Klíčová slova řízení dopravy, autonomní doprava, multi-agent pathfinding, multi-agent pickup and delivery, conflict based search, token passing, simulace

Abstract

This thesis deals with the problem of managing autonomous traffic in cities. We consider future transport system where all vehicles are shared and can transport people or objects on their own without a driver. Customers only create transport requests. The aim of the work is to design and test algorithms for the system that could be used to solve the problem of finding suitable routes for vehicles. Key objectives are the overall energy performance of the system and customer satisfaction. To verify the results of the algorithms, a simulation is implemented which, in a simplified model, mimics traffic and city requirements. First, an environment is defined that describes how that transportation system might work. Three different existing algorithms are considered for traffic management, which differ in the distribution of requirements and the planning of vehicle routes. All three algorithms are modified and improved to allow for different priority requests. The testing is focused on the ability of algorithms to meet requirements at different vehicle to task ratios. Furthermore, it is also examined how long customers waited under certain conditions or what the computing power requirements are for individual algorithms.

Keywords traffic management, autonomous transport, multi-agent pathfinding, multi-agent pickup and delivery, conflict based search, token passing, simulation

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době rychlého technologického růstu často dochází k využívání umělé inteligence pro zvýšení efektivnosti v různých odvětvích. Jedním z odvětví, které naprostá většina lidí využívá každý den, je doprava. Vliv technologického pokroku má v této oblasti vliv především na lepší vozidla, ale nedochází k zefektivnění dopravního systému jako celku. Ten je velmi podobný jako před padesáti lety, avšak v dnešní době jsou na něj mnohem větší nároky.

Možností, jak tento systém vylepšit, je mnoho. Mezi mírnější změny můžeme řadit využití inteligentních semaforů, které mezi sebou dokáží komunikovat. Razantnější úprava by představovala přetvoření celého dopravního systému od základu.

Jednou z nedokonalostí současného řešení jsou samotní řidiči a fakt, že cíl cesty jednotlivých vozidel není znám celému systému. V budoucnosti, kdy budou existovat zcela autonomní vozidla, by ve městech mohli být lidští řidiči zcela nahrazeni umělou inteligencí ve vozidlech. Uživatelé by poté pouze volili svou cílovou destinaci. Pro minimalizaci celkového počtu aut by pro plnění požadavků mohla být využívána sdílená vozidla.

V této situaci nastává problém, jak nejefektivněji centrálně řídit všechna vozidla v dopravním systému pro minimalizaci energetické náročnosti a maximální uživatelskou spokojenost. U řízení autonomní dopravy je potřeba vyřešit, jak se aktuální požadavky rozdělí mezi dostupná vozidla, jakým způsobem budou naplánovány trasy či jak zacházet s nevyužitými vozidly.

Kapitola 1

Cíl práce

Hlavním cílem této práce je navrhnout a otestovat algoritmy pro řízení autonomní dopravy ve městech. Důležitým částí je rešerše problematiky dopravních systémů a také nastudování teoretických algoritmů, které by mohly být aplikovatelné na tento problém. Navržené algoritmy by měly být založené na odlišných principech, aby při testování vynikaly v odlišných oblastech. Pro účely testování je zapotřebí vytvořit simulační program, který se bude snažit napodobit problematiku dopravního systému.

Kapitola 2

Úvod do problematiky

2.1 Dopravní systémy

Dopravní systém je konkrétní dopravní síť s její poptávkou po dopravě a jejími řídícími pravidly[1]. Aby tedy nějaký dopravní systém mohl existovat, je potřeba specifikovat pravidla, která musí být dodržována, a zároveň musí existovat požadavky, které budou dopravním systémem realizovány.

Primárním cílem dopravního systému je přeprava osob či věcí z jednoho místa na druhé. Jsou ale i další cíle jako například:

- zvýšení dopravní bezpečnosti,
- ochrana přírodních zdrojů a zamezení znečištění přírody či
- zvýšení ekonomické efektivity.

Mezi vyšší cíle řadíme tyto:

- zvýšení kvality života,
- zvýšení kvality životního prostředí a
- zvýšení kvality lokální ekonomiky.[2]

Dnešní dopravní systémy mají mnoho nedostatků a problémů, které znemožňují plnit ve vyšší míře cíle výše. Jedním z hlavních diskutovaných témat je v dnešní době ekologie dopravy. Například letecká doprava je nejrychlejší možná a pohodlná pro lidi, avšak velmi znečišťuje životní prostředí. Pokud bychom ji ale chtěli na kratších vzdálenostech nahradit automobilovou přepravou, mohlo by dojít k přehlcení silniční infrastruktury. Proto je balancování mezi druhy přepravy velmi obtížné. Problémy, se kterými se setkáváme u většiny druhů přepravy, jsou například nehodovost, hluk, rychlostní limity, a především absence centrálního řízení dopravy.

V jednom dopravním systému může být jeden, či více druhů dopravních prostředků. Lze se tedy zaměřit například pouze na železniční či automobilovou přepravu, ale lze uvažovat i jejich kombinaci. Mezi druhy dopravy patří například letecká, železniční, vodní či silniční. A právě silniční doprava bude předmětem této práce.

2.1.1 Vývoj řízení dopravy

Řízení silniční dopravy nebylo po dlouhou dobu existence lidstva potřeba řešit. Až s příchodem prvních spalovacích motorů vznikla potřeba se tímto problémem zabývat. Vůbec první semafor byl nainstalován v roce 1868 u budovy parlamentu v Londýně. Jednalo se o přístroj na svítiplyn ovládaný manuálně policistou. Semafor ale byl bezpečnostním rizikem, protože občasně docházelo k jeho explozi a následně zranění obsluhy.

Na počátku 20. století se objevil první elektronický systém pro řízení dopravy, který byl nainstalován v Clevelandu. V roce 1928 Charles Adler Jr. vyvinul první zvukový semafor. Řidiči, kteří přijeli k červenému světlu a chtěli ho změnit, museli svým autem zatroubit. Byl nainstalován v Baltimoru a byl prvním aktivovaným dopravním signálem ve Spojených státech. Sloužil jako základ pro moderní dopravní signály. Adler také vynalezl tlačítko pro chodce, které bylo instalováno v Baltimoru. Jednalo se o první přechod aktivovaný chodcem.

Již v padesátých letech minulého století vznikla počítačem ovládaná křižovatka. Na křižovatkách byla umístěna tlaková deska, takže počítače věděly, že na červeném světle čeká auto. V šedesátých letech bylo celé řízení dopravy pomocí počítačů ještě vylepšeno.[3]

V dnešní době největší problém v efektivnosti dopravních systémů představují semafory, které mají pevný časovač. Zejména proto vznikají ve velkoměstech dopravní zácpy. Budoucnost řízení dopravy na křižovatkách by se mohla stát mnohem chytřejší a efektivnější, jakmile by tyto semafory začaly mluvit mezi sebou a s auty, říká Ali Hajbabaie. Pokud by takový systém existoval mohl by zmenšit zdržení na silnicích o 43 %.[4]

2.1.2 Prvky dopravního systému

Obecně se každý dopravní systém skládá z pěti prvků, a to:

Účastníci silničního provozu Lidé jsou aktivními účastníky dopravního systému. Tento prvek je velmi proměnný a nepředvídatelný. Fyzicky je člověk měřitelný a obvykle klasifikovatelný. Mezi fyzické faktory patří: vnímání, reakční limity či úroveň schopnosti řídit vozidlo. Psychické rozhodování lidí je ovšem obtížné měřit a vyčíslit.

Chování většiny lidí se řídí normálním rozdělením, např. u reakční doby či rychlosti jízdy. Je proto nutné předpokládat oba extrémy.

Kromě lidských řidičů je nutné nezapomínat také na chodce. Jejich pohyb po komunikacích je nutný, ale ztěžuje celý dopravní systém. Chodci mohou přecházet silnici buď mimo vyznačený přechod, kde ale nemají přednost před vozidlem a možný střet s vozidly je pro ně nebezpečný, nebo na vyznačených přechodech bez světelné kontroly. Zde má sice chodec přednost, ale tyto situace nelze dopředu plánovat. Na světelných přechodech je situace nejideálnější, avšak i tak mnoho lidí zákaz vejití do silnice ignoruje, a i s těmito případy je nutné počítat.

Vozidla Vozidla jsou nejdůležitějším prvkem, ale také jsou velmi rozmanitá v mnoha ohledech. V České republice existuje 16 řidičských oprávněni podle typu vozidla. Další různorodost se týká vzhledu vozidla. 20 různých designů vozidla pojme 95% percentil. Jsou zde také velké rozdíly ve zrychlení či zpomalení a možnosti otáčení.

Vozovky Vozovky se dělí na 4 třídy:

1. vozovky s omezeným přístupem – dálnice,
2. tepny,
3. sběrny,
4. místní ulice.

Rozdíl mezi nimi je vidět na obrázku 2.1. Kvalita komunikací a péče o ně povětšinou klesá podle typu.

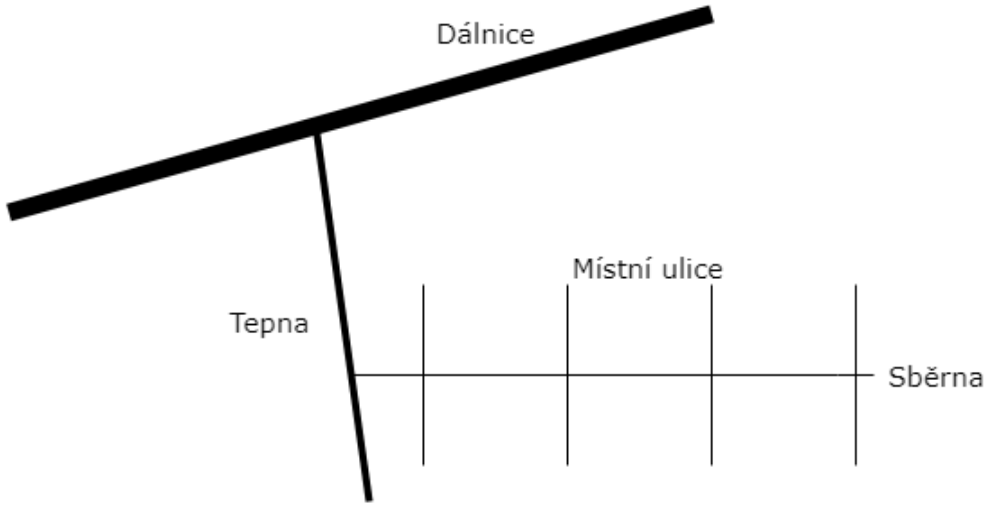
Řídící prvky Možností, jak řídit dopravu, je velké množství. Mezi ty nejdůležitější patří:

- dopravní značení na silnicích (pruhy, přechody, příkazy směru jízdy, ...),
- dopravní značky (zákaz vjezdu, dej přednost jízdě, ...),
- světelné signály (semafory, světla na železničních přejezdech).

Účel řídicích prvků je podpora silniční bezpečnosti a efektivity prostřednictvím zajištění správného pohybu uživatelů na ulicích, silnicích, cyklostezkách a veřejném prostranství. Každý nástroj pro řízení dopravy by měl splňovat tyto body:

- splnit potřebu,
- upoutat pozornost,
- vyjádřit jasnou a jednoduchou zprávu a
- poskytnout dostatečný čas na správnou reakci.

Obecné prostředí – počasí, osvětlení, právní aspekty



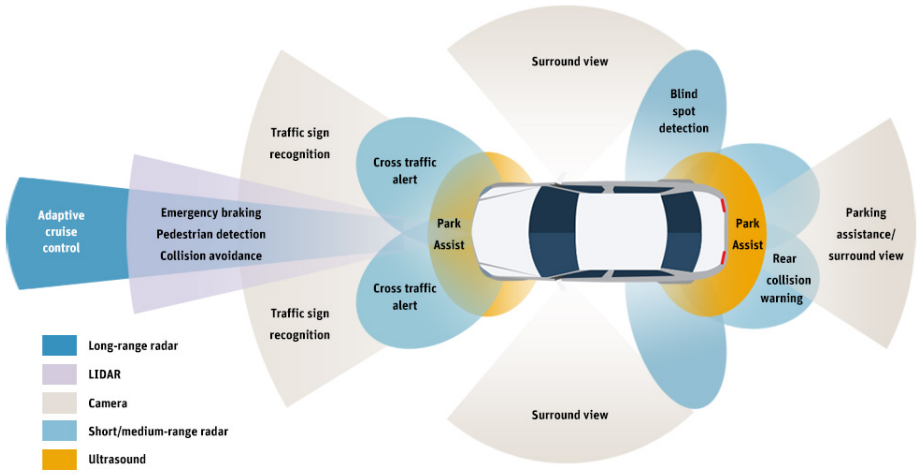
Obrázek 2.1: Typy vozovek

2.2 Autonomní doprava

Autonomní systémy jsou charakterizovány jako systémy schopné činit rozhodnutí nezávisle na lidských zásazích, ale na rozdíl od pouhé automatizace mohou činit tato rozhodnutí, a přitom čelit nejistotě. Autonomní vozidla se spoléhají na umělou inteligenci, senzory a data velkého objemu, aby analyzovala informace, přizpůsobila se měnícím se okolnostem a zvládala složité situace jako náhražku lidského úsudku. Ten by již nebyl potřebný pro konvenční provoz vozidel, tedy střídání jízdních pruhů, parkování, předcházení kolizím a brzdění. Tyto řídící systémy reagují na změny vnějších podmínek mnohem rychleji než běžný lidský řidič a jsou doplněny o komunikaci mezi vozidly a infrastrukturou, což autonomním vozidlům umožňuje učit se od jiných automobilů. Příklad senzorů autonomního vozidla lze vidět na obrázku 2.2.[5]

Aby automobil mohl jezdit bez lidského řidiče uvnitř, některé společnosti používají řízení na dálku, kdy je automobil ovládán člověkem na dálku. Zde se ale nejedná o autonomní řízení, protože automobil nevykonává všechny své funkce nezávisle a dostatečně samostatně. Pokud je závislý na komunikaci nebo spolupráci s externími subjekty je považován spíše za kooperativní než za autonomní.[7]

Vídeňská úmluva o silničním provozu z roku 1968, kterou podepsalo přes 70 zemí světa, zavádí zásady, jimiž se řídí dopravní předpisy. Jedním ze základních principů úmluvy byl koncept, že řidič má vždy plnou kontrolu a je zodpovědný za chování vozidla v provozu.[8] Pokrok technologie, která pomáhá řidiči a přebírá jeho funkce, tento princip podkopává, což znamená, že velká část základů musí být přepsána. V současné době je na území Evropy povoleno testovat autonomní vozidla ve Velké Británii, Francii, Švýcarsku a Maďarsku. Také další státy postupně připravují potřebné zákony. Spojené státy ame-



Obrázek 2.2: Příklad senzorů autonomního automobilu[6]

rické, které nepodepsaly Vídeňskou úmluvu o silničním provozu, nepočítají s velkou autonomitou vozidel, ale nutně ji nezakazují. Jednotlivé členské státy postupně přijímají zákony pro tato vozidla, a to především po první smrtelné nehodě, která se stala na Floridě v roce 2016, u níž došlo ke střetu vozidla řízeného autopilotem Tesla a 18-kolovým tahačem.

Inovativní technologie, jako jsou autonomní vozidla, vytvářejí rizika a nezamýšlené důsledky, které mohou snížit akceptaci společností. Příkladem jsou environmentální rizika, tržní rizika, sociální rizika, organizační rizika, politická rizika, finanční rizika, technologická rizika a rizika turbulencí. S autonomními vozidly je spojováno pět typů technologických rizik: bezpečnost, odpovědnost, soukromí, kybernetická bezpečnost a vliv průmyslu. Pro zajištění, že společnost vytěží maximum ze vznikajícího trhu s autonomními vozidly, je prvořadé, aby vlády zavedly nová opatření a předpisy pro řízení rizik spojených s autonomními vozidly.

Odpovědnost za samořiditelná auta je rozvíjející se oblastí práva a politiky, která určí, kdo je odpovědný, když automatizovaný automobil způsobí fyzickou škodu osobám nebo poruší pravidla silničního provozu. Až dojde k přechodu od automatizovaných aut řízených člověkem na autonomní automobily, bude zapotřebí zajistit vývoj stávajících právních předpisů o odpovědnosti tak, aby spravedlivě identifikovaly strany odpovědné za škody a úrazy, a řešily možnost střetu zájmů mezi lidmi, provozovatelem systému, pojišťovateli a veřejnou pokladnou[9].

2.2.1 Úrovně autonomity

Autonomní vozidla jsou řazena do různých kategorií na základě jejich vlastností. Society of Automotive Engineers provádí toto zařazování dle šesti úrovní

automatizace.

0. úroveň – No Automation Všechny aspekty jízdy závisí čistě na řidiči, a to i v případě, že je auto vybaveno výstražnými nebo zásahovými systémy.

1. úroveň – Driver Assistance Automobil je vybaven specifickými jízdními asistenty, které umožňují například držení automobilu v jízdním pruhu, adaptivní tempomat či brzdění.

2. úroveň – Partial Automation Asistenční systémy automobilu jsou schopné jak zatáčet, tak i zrychlovat a zpomalovat. Očekává se, že lidský řidič provede zbývající aspekty dynamického jízdního úkolu.

3. úroveň – Conditional Automation Jízdní systém automobilu je plně schopen provádět všechny jízdní aspekty. Je předpokládáno, že lidský řidič je schopen kdykoliv zasáhnout, pokud je to zapotřebí.

4. úroveň – High Automation Automobil je plně schopen provádět všechny aspekty jízdy, i když lidský řidič nereaguje na žádosti o zasáhnutí.

5. úroveň – Full Automation Jízdní systém automobilu je schopen plnit všechny aspekty dynamické jízdy na všech vozovkách a v prostředí, kde může řídit lidský řidič.

Od 3. úrovně výše se jedná o automatizované řízení automobilu. Od 4. úrovně pak lze mluvit o autonomním řízení.[7]

Mezi nejznámější samořiditelná auta patří Tesla, která v jejich vozidlech nabízí 2. úroveň autonomity. Na tuto úroveň se řadí také autopiloti od společností Cadillac, Mercedes-Benz či Volvo. Společnosti Audi a Waymo začali testovat automobily se 3. úrovní již v roce 2017, ale celkový proces brzdí zákony a nevyřešené otázky odpovědnosti.

2.2.2 Budoucnost autonomní dopravy

V současné době není automobilová doprava nijak centrálně ani distribuovaně řízena. Vozidla mezi sebou nekomunikují a o trase rozhoduje řidič. Pokud budou v budoucnosti na silnicích jezdit autonomní automobily, nabízí se otázka, zda by pro celý dopravní systém nebylo výhodnější, aby existoval centrální systém pro plánování tras všech vozidel na silnicích. V praxi by zákazník pouze nasedl do auta, vybral si cílovou destinaci a o zvolené trase by nerozhodoval. Systém by tento požadavek přijal a s ohledem na ostatní subjekty na komunikacích by vyhodnotil ideální trasu jak pro uživatele, tak pro celý systém.

V dopravním systému si nejsou všichni účastníci rovni. Příkladem jsou vozidla integrovaného záchranného systému, tedy vozidla zdravotní služby,

hasiči či policie. Tyto instituce mají přednostní požadavky na přepravu a nový systém by je mohl respektovat lépe než ten současný. Pokud se někde stane dopravní nehoda, nejen že to mohou autonomní vozidla spolehlivě nahlásit, ale plánovací systém bude počítat s tím, že v tomto místě je silnice neprůjezdná a odkloní ostatní automobily na jinou trasu. Zároveň poskytne integrovanému záchrannému systému nejrychlejší možnou cestu.

V dnešní době ve vyspělých státech je na 1000 obyvatel v průměru 500 automobilů (Česká republika – 485, USA – 797)[10]. Mnoho lidí využívá svůj automobil jen jako dopravní prostředek do práce a domů. Jiní lidé jezdí do své práce veřejnou hromadnou dopravou a auto využívají jen ve svém volném čase. Z tohoto důvodu existuje zbytečně až moc velký počet automobilů. Pokud by byla v budoucnosti zavedena sdílená autonomní přeprava, bylo by cestování levnější, protože by se vozidla efektivněji využívala. Sdílená auta by sama jezdila po městech a plnila by účel podobně jako dnes taxi služby. V dopravních špičkách by přesto musel jezdit velký počet vozidel, aby bylo zajištěno plnění požadavků zákazníků, ale tento objem by mohl být o desítky procent menší než v současné době.

Kapitola 3

Teoretický základ

3.1 A*

A* je algoritmus používaný pro vyhledávání optimální cesty v kladně ohodnocených grafech. Vznikl v šedesátých letech minulého století vylepšením Dijkstrova algoritmu pomocí heuristiky. Původní verze nesla název A1, vylepšená A2, až při důkazu optimality v roce 1968 ho Bertram Raphael[11] pojmenovat A*, kde jsou ukryty všechny jeho možné verze.

A* používá hladový princip pro nalezení optimální cesty z daného počátečního uzlu do požadovaného cílového. Princip je podobný prohledávání do šířky, kde dochází k postupné expanzi uzlů od počátečního, dokud není splněno kritérium ukončení. Při každé iteraci musí hlavní smyčky určit, kterou ze svých cest prodloužit. To dělá na základě ceny současné cesty a odhadované ceny až k cíli. Konkrétně A* vybere cestu, která minimalizuje tuto funkci:

$$f(n) = g(n) + h(n) \tag{3.1}$$

, kde n je další uzel v grafu, $g(n)$ je cena cesty z počátečního uzlu do n a $h(n)$ je heuristika odhadující cenu cesty z n do cílového uzlu.

Heuristická funkce je specifická pro každý problém. Pokud je vybrána správně vybrána a nenadhodnocuje cenu cesty nad není skutečnou hodnotu, tak A* garantuje nalezení cesty s nejnižší cenou. Například pokud je problém definován nad 2D mřížkou, tak dobrou volbou pro odhad $h(n)$ je euklidovská vzdálenost.

Na nezáporně ohodnocených grafech je zaručena jeho úplnost a konečnost. Při správně zvolené heuristické funkci můžeme vždy očekávat optimální řešení.

Algoritmus 1: A*

```
Input  : start, goal(n), h(n), expand(n)
Output: path
1  if goal(start) = true then
2  |  return makePath(start)
3
4  open ← start
5  closed ← ∅
6  while open ≠ ∅ do
7  |  sort(open)
8  |  n ← open.pop()
9  |  kids ← expand(n)
10 |  foreach kid ∈ kids do
11 |  |  kid.f ← (n.g + 1) + h(kid)
12 |  |  if goal(kid) = true then
13 |  |  |  return makePath(kid)
14 |  |  if kid ∩ closed = ∅ then
15 |  |  |  open ← kid
16 |  closed ← n
17 return ∅
```

3.2 Multi-Agent Path-Finding

Multi-agentní hledání cest se zabývá problémem, kdy pro více jak jednoho agenta najednou je potřeba najít optimální trasu ze startovní pozice do cílové a zároveň se agenti nesmí při cestě srazit.

Vstupem do tohoto problému s k agenty je trojice (G, s, t), kde $G = (V, E$ je neorientovaný graf, $s : [1, ..., k] \to V$ mapuje agenty na počáteční uzly a $t : [1, ..., k] \to V$ mapuje agenty na cílové uzly. Předpokládá se diskrétnost času a v každém kroku je každý agent umístěn v jedné z vrcholů grafu a může provést jednu akci. Akce je funkce $a : V \to V$, kde $a(v) = v'$ znamená, že pokud agent je v uzlu v a provede a, poté v dalším časovém kroku bude v uzlu v'. Každý agent má dva typy akce: pohyb nebo čekání na místě. Pohyb znamená, že se agent přesune ze současného vrcholu v přesune na sousední vrchol v' v grafu.[12]

Plán je posloupnost akcí vedoucích agenta z počátečního umístění do cíle. Délka plánu (pro agenta) je definována časem, kdy agent dosáhne svého cíle a už ho neopouští. Cílem multi-agentního hledání cest je najít plány pro všechny agenty, aby mezi sebou nekolidovaly v čase ani prostoru. Ne každý validní plán je ale optimální. Typicky se minimalizují 2 možná kritéria:

Makespan – doba mezi začátkem prvního agenta a dokončením posledního, tedy celkový čas.

Sum of costs – suma délek plánů všech agentů.

Cílem MAPF řešiče je najít řešení, tedy množinu plánů pro každého agenta, která které můžou být provedeny bez kolize. Existuje několik druhů kolizí, které se odvíjejí od prostředí systému. Nechť π_i a π_j je pár plánů agentů.

Uzlový konflikt – Uzlový konflikt nastane mezi π_i a π_j právě tehdy, když v obou plánech je obsazen stejný uzel ve stejném čase, tedy $\pi_i[x] = \pi_j[x]$.

Hranový konflikt – K hranovému konfliktu mezi π_i a π_j dojde v případě, že oba agenti plánují ve stejný čas použít ve stejném směru stejnou hranu, tedy $\pi_i[x] = \pi_j[x]$ a $\pi_i[x+1] = \pi_j[x+1]$.

Pronásledovací konflikt – Tento konflikt vznikne právě tehdy, když jeden agent plánuje obsadit uzel, kde v minulém kroku byl jiný agent, tedy $\pi_i[x+1] = \pi_j[x]$.

Vyměňovací konflikt – Vyměňovací konflikt nastane v případě, když si dva agenti vymění své pozice, tedy $\pi_i[x+1] = \pi_j[x]$ a $\pi_i[x] = \pi_j[x+1]$.

Existuje více druhů konfliktů a konkrétní zakázané konflikty jsou vždy definovány podle instance problému.[12]

Algoritmy pro řešení problému multi-agentního hledání cest lze obecně rozdělit do dvou kategorií: coupled a decoupled. Coupled přístup řeší plány všech agentů najednou, zatímco decoupled rozkládá problém do podproblémů, obvykle nalezením plánů pro každého agenta zvlášť a následně jsou řešeny výsledné kolize.

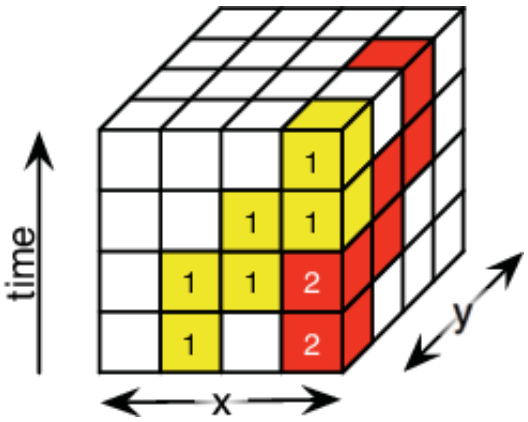
K řešení tohoto problému lze přistoupit dvěma způsoby:

Techniky založené na prohledávání – prohledávání stavového prostoru (CBS[13], ICTS[14])

Techniky založené na redukci – převedení problému na jiný(SAT, CSP, ASP, ...)

Multi-agentní hledání optimální cesty je NP-těžký problém, protože je to zobecnění problému posouvání $n^2 - 1$ dílů puzzle[15]. Řešiče tohoto problému proto mohou být optimální či suboptimální a úplné nebo neúplné. Důležité proto bylo najít algoritmus, který bude optimální a zároveň úplný.

První pokus vyřešit tento problém byla práce Cooperative A*[16]. Plány zde byly vytvářeny pro každého agenta zvlášť a cílem bylo vyhnout se kolizi s dříve naplánovaných agentem. Byl zde použit koncept takzvaný "reservation table". Jedná se o graf v čase, kde si každý agent rezervuje uzel v daném čase. Pokud má graf například podobu 2D mřížky, tak s rezervacemi v čase vznikne 3D kvádr, viz. obrázek 3.1.



Obrázek 3.1: Reservation table[17]

Toto řešení má ovšem svoje omezení, a ne vždy dává úplné a optimální řešení.

Přístupy vedoucí k optimálnímu řešiči nejprve vedle přes prohledávání k-agentového stavového prostoru[18] a později se objevily další možnosti, jako například Conflict-Based Search[13], kde se využívá více úrovňový algoritmus na řešení kolizí v plánech.

3.2.1 Conflict-Based Search

Conflict-Based Search(CBS)[13] je dvojúrovňový algoritmus pro řešení problému multi-agentního hledání cest. CBS zaručuje optimální řešení jako většina coupled přístupů, ale hledání cest, které CBS provádí, je striktně po jednom agentovi, podobně jako u decoupled přístupů. Kromě optimality se vyznačuje úplností a jeho výpočetní složitost roste exponenciálně v počtu konfliktů.

Stavový prostor multi-agentního hledání cest je exponenciální v počtu agentů. Naproti tomu stavový prostor u hledání cesty pro jednoho agenta je pouze lineární ve velikosti grafu. CBS pro řešení používá dekompozici problému s více agenty na velké množství problémů s jedním agentem. Každý z nich je relativně jednoduchý na řešení, ale těchto problémů může být exponenciálně mnoho.

CBS používá pojem cesta jen v kontextu jediného agenta a pojem řešení používá k označení sady k cest pro danou sadu k agentů.

Omezení pro agenta a_i je trojice (a_i, v, t), kde agentovi a_i je zakázáno být v čase t v uzlu v. V průběhu algoritmu je agentům postupně přidávána další omezení. Konzistentní cesta pro agenta a_i je cesta, která splňuje všechna jeho omezení. Podobně konzistentní řešení je řešení, které je tvořeno z cest, kde cesta pro agenta a_i je v souladu s jeho omezeními.

Konflikt je čtveřice (a_i, a_j, v, t), kde agenti a_i a a_j obsazují uzel v v čase t. Řešení je validní, pokud žádná z jejich cest nemá žádný konflikt. Konzistentní řešení může být neplatné, pokud navzdory skutečnosti, že cesty jsou konzistentní s omezeními jejich jednotlivých agentů, stále existují konflikty.

Klíčovou myšlenkou CBS je rozrůstat množinu omezení pro každého z agentů a najít cesty, které jsou v souladu s těmito omezeními. Pokud tyto cesty mají konflikty a jsou tedy neplatné, konflikty se vyřeší přidáním nových omezení.

CBS funguje ve dvou úrovních. Na vyšší úrovni se objevují nové konflikty a přidávají se omezení. Nízká úroveň aktualizuje cesty agentů tak, aby byly konzistentní s novými omezeními.

Na vyšší úrovni CBS prohledává takzvaný constraint tree (CT), neboli strom omezení. CT je binární a každý jeho uzel N obsahuje následujicí data:

1. **Množina omezení** – Kořen CT má tuto množinu prázdnou. Potomci ve stromu dědí omezení jejich rodičů a zároveň přidávají svoje pro jednoho agenta.

2. **Řešení** – Množina cest, jedna pro každého agenta. Cesta pro agenta a_i musí být konzistentní s jeho omezeními. Takové cesty jsou získávány nižší úrovní algoritmu.

3. **Celková cena řešení** – Cena daného řešení může určena různými způsoby – například suma délek všech cest v řešení.

Uzel v CT je cílový, pokud jeho řešení je validní. Vyšší úroveň algoritmu provádí prohledávání CT do šířky, kde jsou uzly seřazeny podle jejich ceny řešení.

Algoritmus 2: high-level of CBS

```
   Input: MAPF instance
1  R.constraints = ∅
2  R.solution = find individual paths using the low-level()
3  R.cost = SIC(R.solution)
4  insert R to OPEN
5  while OPEN not empty do
6  |  P ← best node from OPEN
7  |  Validate the paths in P until a conflict occurs.
8  |  if P has no conflict then
9  |  |  return P.solution
10 |  C ← first conflict (a_i, a_j, v, t) in P
11 |  foreach agent a_i in C do
12 |  |  A ← new node
13 |  |  A.constraints ← P.constraints + (a_i, s, t)
14 |  |  A.solution ← P.solution
15 |  |  Update A.solution by invoking low-level(a_i)
16 |  |  A.cost = SIC(A.solution)
17 |  |  Insert A to OPEN
```

Při zpracování uzlu se na základě omezení vyvolá nižší úroveň algoritmu. Toto vyhledávání vrátí nejkratší cesty pro všechny agenty vzhledem k jejich omezení. Jakmile je u každého agenta nalezena konzistentní cesta s ohledem na jeho omezení, jsou tyto cesty validovány s ohledem na ostatní agenty. Ověření validnosti řešení probíhá postupně pro všechny cesty. Pokud všichni agenti dosáhnou svého cíle bez jakéhokoli konfliktu, je tento uzel CT deklarován jako cílový uzel a je vráceno aktuální řešení, které obsahuje tuto sadu cest. Pokud bude při validaci nalezen konflikt dvou či více agentů, validace se zastaví a uzel nebude prohlášen jako cílový.

Pro vyřešení nalezeného konfliktu je zapotřebí vytvořit nová omezení, aby nedocházelo ke střetu agentů. Pro dosažení optimality, je zapotřebí vytvořit obě verze omezení pro každého agenta a toto budou potomci současného uzlu v CT. Při konfliktu (a_i, a_j, v, t) tedy vznikne omezení (a_i, v, t) pro levého potomka a (a_j, v, t) pro pravého.

Může nastát situace, že dojde ke konfliktu více než dvou agentů. Existují dva přístupy, jak tento problém vyřešit. Při konfliktu k agentů je vytvořeno k potomků, kde každému vznikne $k - 1$ omezení. Tím pádem nemůže platit předpoklad binarity CT. Druhou ekvivalentní možností je zaměřit se pouze na první dva agenty v konfliktu. Zde bude dosažena binarita, ale potomci nebudou cílovým uzlem, protože se u nich projeví zbývající konflikty.

V nižší úrovni algoritmu je dán agent (a_i a přidružená množina omezení. Na grafu je prováděno vyhledávání optimální cesty, která splňuje všechna omezení. Všichni ostatní agenti jsou ignorování. Vyhledávání probíhá kromě standardních dvou dimenzí také v časové dimenzi. Volba vyhledávacího algo-

ritmu pro jednoho agenta může být libovolná. Dobrou volbou je použití A* se správnou heuristikou.

CBS je optimální řešič problému multi-agentního hledání cest a je ekvivalentní k algoritmech založených na prohledávání stavového prostoru pomocí A*. Důležité je jako má graf podobu. Pokud se v něm objevují více zúžených míst, tak CBS má výhodu nad přístupy pomocí A*. V případě hodně prostoru je situace opačná. [13]

3.3 Multi-Agent Pickup and Delivery

Multi-Agent Pickup and Delivery(MAPD)[19], neboli multi-agentní vyzvednutí a doručení, je rozšíření problému multi-agentního hledání cest. V reálném prostředí, jako jsou například automatizované sklady, nelze předpokládat, že každý agent má svoji cílovou pozici a všichni počkají na dokončení všech úkolů. V MAPD problému se agenti musí vypořádat s nepřetržitým proudem úkolů, které vznikají dynamicky v čase. Celkový počet úkolů tedy může libovolný, na rozdíl od původního problému. Jeden agent přitom může být přidělen jen na jeden úkol. Nejprve je nutné, aby dorazil na vyzvedávají pozici úkolů a poté doručovací pozici. Vždy je nutné se vyhýbat kolizím s ostatními agenty.

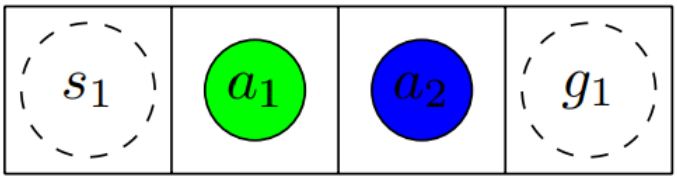
Formálně se instance MAPD problému skládá z m agentů $A = \{a_1, a_2, ..., a_m\}$ a neorientovaného grafu $G = (V, E)$, jehož uzly odpovídají pozicím a jeho hrany cestám, po kterých můžou agenti cestovat. Nechť označme $l_i(t)$ jako lokaci agenta a_i v diskrétním čase t. Agent a_i má svou počáteční lokaci označenou jako $l_i(0)$. V každém čase t agent buď zůstane ve svém aktuálním umístění $l_i(t)$ nebo se přesune na sousední pozici, tedy $l_i(t+1) = l_i(t)$ nebo $(l_i(t), (l_i(t+1)) \in E$.

Agenti se musí vyhýbat kolizím s ostatními agenty, tedy: dva agenti nesmí být v jeden čas ve stejné lokaci a zároveň se dva agenti nesmí pohybovat po stejné hraně v opačných směrech ve stejný čas. Tedy pro všechny agenty a_i a a_j, pokud $a_i \neq a_j$, a čas t platí: $l_i(t) \neq l_j(t+1)$ nebo $l_j(t) \neq l_i(t+1)$.

Uvažujme sadu úkolů τ, která obsahuje množinu všech neprovedených úkolů. V každém čase může systém přidat nový úkol do sady τ. Každý úkol τ_t je určen vyzvedávací lokací $s_t \in V$ a doručovací lokací $g_t \in V$. Agent se nazývá volným, pokud v daném čase neprovádí žádný úkol. V opačném případě je obsazený. Volnému agentovi může být přiřazen jakýkoliv úkol $\tau_t \in \tau$. Ten se pak musí přesunout ze své současné pozice na pozici s_t a dokončit úkol na pozici g_t. V moment, kdy agent dorazí na vyzvedávací pozici s_t, úkol je odstraněn ze sady úkolů τ. Po doručení se agent stává znovu volným.

Cílem systému je doručit úkoly co nejrychleji. V důsledku toho je efektivita algoritmu MAPD hodnocena průměrným počtem časových kroků, tzv. servisní dobou, potřebných k dokončení provádění jednotlivých úkolů po jejich přidání do sady úkolů. Algoritmus vyřeší instanci MAPD, pokud je výsledný servisní čas všech úloh ohraničen.

Ne každá instance MAPD je řešitelná. Příklad je vidět na obrázku 3.2, kde agent a_1 ani a_2 nemůže dokončit úkol τ_1.



Obrázek 3.2: Neřešitelná MAPD instance

Agenti by měli mít možnost odpočívat pouze v lokacích, tzv. endpointech, kde nemohou blokovat jiné agenty při plnění jejich úkolů. Množina endpointů V_{ep} obsahuje všechny počáteční pozice agentů, všechny vyzvedávací a doručovací lokace a možné parkovací pozice. Nechť V_{tsk} označmě jako množinu všech možných vyzvedávacích a doručovacích pozic, které nazýváme úkolové endpointy. Množina $V_{ep} \setminus V_{tsk}$ se nazývá množina neúkolových endpointů.

Instance MAPD je řešitelná právě tehdy, když:

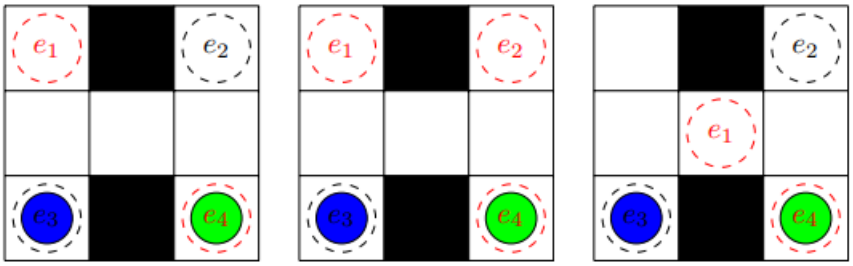
1. celkový počet úkolů je konečný,
2. není zde menší počet neúkolových endpointů než agentů,
3. pro jakékoliv dva endpoity existuje cesta, která je spojuje a zároveň neprochází jiným endpointem.

Řešitelná MAPD instance (s alespoň jedním úkolem) má nejméně $m + 1$ endpointů. Obrázek 3.3 ukazuje tři příklady instancí MAPD. Levá instance je řešitelná. Prostřední není řešitelná, protože zde jsou dva agenti a pouze jeden neúkolový endpoint. Poslední příklad není řešitelný, protože například všechny možné cesty mezi endpointy e_2 a e_3 vedou přes endpoint e_1.[19]

3.3.1 Token Passing

Token Passing(TP)[19] je decoupled algoritmus pro řešení MAPD problému. Jeho idea je založena na podobné myšlence jako práce Cooperative A*[16], kde agenti plánují své cesty jeden po druhým. Pro řešitelné MAPD instance vždy vrací řešení.

Plánování zde funguje na principu předávání *tokenu* a díky tomu lze celý algoritmus rozšířit do plně distribuované podoby. Díky tomu odpadá nutnost mít všechen výpočetní výkon soustředěný na jednom serveru a díky vzájemné komunikaci agentů může každý z nich provádět výpočty sám.



Obrázek 3.3: Tři MAPD instance. Modrý a zelený kruh jsou agenti. Přerušované červené kruhy jsou úkolové endpointy a černě přerušované kruhy jsou neúkolové endpointy.[19]

Množina úkolů zde obsahuje všechny úkoly, které nejsou přiřazeny žádnému agentovi. *Token* zde představuje synchronizovanou sdílenou paměť, která obsahuje všechny současné naplánované cesty, množinu úkolů a přiřazení agentů k jednotlivým úkolům.

Algoritmus 3: Token Passing

```
1  Initialize token with the (trivial) path [loc(a_i)] for each agent a_i
2  while true do
3      Add all new tasks, if any, to the task set τ
4      while agent a_i exists that requests token do
5          /* system sends token to a_i - a_i executes now */
6          τ' ← { τ_j ∈ τ| other path in token ends in s_j or g_j}
7          if τ' ≠ ∅ then
8              t ← arg min_{τ_j ∈ τ'} h(loc(a_i), s_j)
9              Assign a_i to t
10             Remove t from τ
11             Update a_i's path in token with Path1(a_i, τ, token)
12         else if no task τ_j ∈ τ exists with g_j = loc(a_i) then
13             Update a_i's path in token with the path [loc(a_i)]
14         else
15             Update a_i's path in token with Path2(a_i, token)
16         /* a_i returns token to system - system executes now */
17     All agents move along their paths in token for one timestep
18     /* system advances to the next timestep */
```

Pseudokód 3 ukazuje kód algoritmu TP, kde $loc(a_i)$ označuje současnou pozici agenta a_i. Agent a_i hledá všechny cesty pomocí algoritmu A* prohledáváním stavového prostoru, kde stav je dvojice pozice a čas. Orientovaná hrana grafu existuje ze stavu (l, t) do stavu $(l', t + 1)$ právě tehdy, když $l = l'$

nebo $(l, l') \in E$. Stav (l, t) je odstraněn ze stavového prostoru právě tehdy, když a_i se nachází na pozici l v čase t, aby se zabránilo kolizi s ostatními agenty, kteří se pohybují po jejích cestě. Podobně tak hrana ze stavu (l, t) do stavu $(l', t+1)$ je odstraněna ze stavového prostoru právě tehdy, když se agent a_i pohybuje z pozice l na pozici l' v čase t.

Protože je potřeba hledat pouze cesty mezi endpointy, ceny cest ze všech pozic do všech endpointů mohou být předpočítány a pak použity jako h-hodnota ve vyhledávání A*.

TP pracuje následovně. Systém inicializuje *token* prázdnými cestami, protože všichni agenti jsou na jejich počáteční pozici. V každém časovém okamžiku může systém do množiny úkolů přidat nový úkol. Každý agent, který dokončil jeho cestu v *tokenu*, může o něj požádat jednou za časový krok. Systém poté postupně pošle *token* všem agentům, kteří o něj požádali. Při přiřazení *tokenu*, agent si vybere úkol, jehož vyzvedávací ani doručovací pozice není v cestě žádného jiného agenta.

- Pokud existuje pro agenta alespoň jeden takový úkol, přiřadí se mu takový, jehož h-hodnota je mezi agentovou pozicí a vyzvedávací pozicí úkolu nejmenší. Agent pat zavolá funkci Path1 pro změnu jeho cesty v *tokenu*, která bude minimalizovat cenu k vyzvedávací pozici úkolu a zároveň nebude kolidovat s již naplánovanými cestami v *tokenu*.

- Pokud žádný takový úkol není, tak mu nebude přiřazen nový úkol v tento časový okamžik. Je důležité zkontrolovat, zda agentovo čekání neblokuje jiného agenta v doučení úkolu. Pokud se agent bez úkolu nachází na doručovací pozici jakéhokoliv úkolu, je zavolána funkce Path2, která naplánuje jeho cestu na nejbližší endpoint, kde nebude blokovat ostatní. Pokud agent nepřekáží ostatním, tak se jeho čekání upraví v *tokenu*.

Nakonec agent vrátí *token* do systému a posune se podle jeho cesty.[19]

3.3.2 Token Passing with Task Swaps

Token Passing with Task Swaps(TPTS)[19] je vylepšená verze algoritmu TP. Důležitou změnou je, že množina úkolů, která u TP obsahovala pouze úkoly nepřiřazené žádnému agentovi, nyní obsahuje všechny úkoly, které ještě nikdo nezačal provádět. Z toho plyne, že agent, který má *token*, tak si může přiřadit nikým nepřiřazený úkol, ale také i úkol, k němuž již je na cestě jiný agent. Pokud si nový agent přiřadí takový úkol, protože je k vyzvedávací blíže, tak původnímu agentovi se přiřazení zruší. Poté je potřeba poslat *token* původnímu vlastníku úkolu, aby se pokusil přiřadit si jiný úkol.

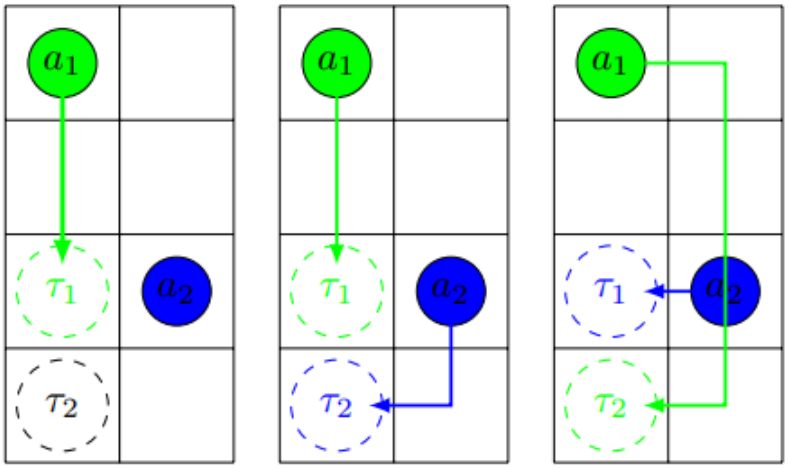
Algoritmus 4 ukazuje pseudokód TPTS. Hlavní smyčka je podobná algoritmu TP a používá i stejné funkce Path1 a Path2. Je zde přidaná funkce $GetTask$, která se pokouší agentovi a_i přiřadit úkol z množiny úkolů τ a najít

Algoritmus 4: Token Passing with Task Swaps

```
1  Initialize token with the (trivial) path [loc(a_i)] for each agent a_i
2  while true do
3      Add all new tasks, if any, to the task set τ
4      while agent a_i exists that requests token do
5          /* system sends token to a_i - a_i executes now */
6          GetTask(a_i, token)
7          /* a_i returns token to system - system executes now */
8      All agents move along their paths in token for one timestep and
         remove tasks from τ when they start to execute them
9      /* system advances to the next timestep */
10 Function GetTask(a_i, token):
11     τ' ← { τ_j ∈ τ | other path in token ends in s_j or g_j }
12     while τ' ≠ ∅ do
13         t ← arg min_{τ_j ∈ τ'} h(loc(a_i), s_j)
14         Remove t from τ'
15         if no agent is assigned to τ then
16             Assign a_i to τ
17             Update a_i's path in token with Path1(a_i, τ, token)
18             return true
19         else
20             Remember token, task set, and agent assignments
21             a'_i ← agent that is assigned to τ
22             Unassign a'_i from τ and assign a_i to τ
23             Remove a'_i's path from token; Path1(a_i, τ, token)
24             Compare when a_i reaches s_j on its path in token to when
                 a'_i reaches s'_j on its path in token'
25             if a_i reaches s_j earlier than a'_i then
26                 /* a_i sends token to a'_i - a'_i executes now */
27                 success ← GetTask(a'_i, token)
28                 /* a'_i sends token to a_i - a_i executes now */
29                 if success then
30                     return true
31             Restore token, task set, and agent assignments
32     if loc(a_i) is not an endpoint then
33         Update a_i's path in token with Path2(a_i, token)
34         if path was found then
35             return true
36     else
37         if no task τ_j ∈ τ exists with g_j = loc(a_i) then
38             Update a_i's path in token with the path [loc(a_i)]
39         else
40             Update a_i's path in token with Path2(a_i, token)
41         return true
42     return false
```

nejkratší cestu do daného endpointu. Pokud je přiřazení a nalezení úspěšné, funkce vrací *true*, v opačném případě $false$.

Při běhu funkce $GetTask$, agent $a + i$ zvažuje všechny úkoly z množiny τ, jejíž vyzvedávací ani doručovací pozice neležení na některé naplánované trase v *tokenu*. Dostupné úkoly prochází postupně podle h-hodnoty z jeho pozice na vyzvedávací pozici úkolu. Pokud nalezne úkol, ke kterému nikdo není přiřazen, tak si ho přisvojí, naplánovanou trasu uloží do *tokenu* a vrátí úspěch. Pokud k danému úkolu je přiřazen agent, porovná délku jejich trasy k vyzvedávací pozici a v případě, že je ho kratší, tak si úkol přiřadí a nad druhým agentem spustí funkce $GetTask$, aby si zajistil jiný úkol. Pokud původní agent nenalezne žádný vhodný úkol, provedu kontrolu, zda nestojí na nějakém endpointu a v případě potřeby se posune.



Obrázek 3.4: Tři MAPD instance. Modrý a zelený kruh jsou agenti. Přerušované kruhy jsou vyzvedávací pozice úkolů.[19]

Obrázek 3.4 ukazuje, že algoritmus TP ani TPTS nemusí optimálně přiřazovat agentům úkoly, aby celková cena řešení byla minimální. Pokud by algoritmus TP začal u modrého agenta a_2, přiřadil by mu úkol τ_1 a zelenému agentovi a_1 úkol τ_2(možnost vpravo). V případě, že by začal u zeleného agenta a_1, tak se úkoly přiřadí optimálně a celková cena řešení bude 4(prostřední možnost). U algoritmu TPTS si modrý agent a_2 vždy vynutí úkol τ_1 a cena řešení bude 6.

3.3.3 Lifelong Centralized Pickup and Delivery

Tento centralizovaný algoritmus byl použit v práci Lifelong multi-agent path finding for online pickup and delivery tasks[19] pro porovnání efektivity a účinnosti oproti algoritmům TP a TPTS, jejichž výpočty jsou decentralizované.

Hlavní myšlenkou tohoto algoritmu je, že v každém časovém bodě se nejprve všem agentům přiřadí jejich endpointy, kam mají směřovat a výsledná MAPF instance se vyřeší nějakým centralizovaným MAPF algoritmem – v tomto případě je použito CBS. Po posunu agentů probíhá znovu přiřazování.

1. Přiřazení agentů V první řadě algoritmus projde všechny volné agenty, kteří stojí na vyzvedávací pozici nějakého z úkolů. Pokud doručovací pozice tohoto úkolu není obsazena jiným agentem, přiřadí systém agentovi tento úkol a jako cílovou pozici mu nastaví doručovací pozici úkolu. Poté je každému volnému agentovi přiřazen buď úkol nebo endpoint jako parkovací pozice. Aby byla výsledná instance řešitelná, musí být každému agentovi přiřazen jiný endpoint. Je tedy důležité, aby vyzvedávací ani doručovací pozice nebyla obsazena jiným agentem. Aby toto bylo dosaženo, postupuje algoritmus tímto způsobem.

Nejprve sestaví množinu možných endpointů X pro volné agenty. Algoritmus hladově sestrojí podmnožinu nepřidělených úkolů τ' začínajíce s žádným. Postupně prochází volné úkoly a pokud jeho vyzvedávací a doručovací pozice se nepřekrývá s nějakým již plněným úkolem nebo úkolem v množině τ', tak tento úkol přidá do τ'. Jako množina X je určen soubor vyzvedávacích pozic z τ'. Pokud je počet volných agentů větší než $|X|$, tak je nutné přidat některé endpointy jako parkovací pozice. V tuto chvíli nelze určit, kteří agenti budou tyto pozice potřebovat, tak je vhodné přidat parkovací místo pro každého agenta.

Poté algoritmus přiřadí každému volnému agentovi úkol. Je zde použita tzv. Hungarian Method[20] s ohodnocovací funkce $c'(a_i, e)$ pro každou dvojici volného agenta a_i a endpoint e. Proměnná c zde udává počet volných agentů, C je dostatečně velká konstanta(větší než jakákoliv možná cena) a funkce $c(a_i, e)$ udává minimální cenu cesty mezi a_i a e. Pokud e je vyzvedávací pozice, tak rovnice takto:

$$c'(a_i, e) = c * C * c(a_i, e) \tag{3.2}$$

V opačném případě je použita tato rovnice:

$$c'(a_i, e) = c * C^2 + c(a_i, e) \tag{3.3}$$

Na základě těchto rovnic jsou volným agentům přiděleny cílové endpointy z X.

2. Plánování cest Algoritmus používá optimální efektivní algoritmus pro řešení MAPF problému CBS. Naplánované cesty minimalizují celkový počet časový kroků(makespan).

Plánování cest lze zefektivnit tak, že se první naplánují cesty pro agenty již plnící úkol a v druhé řadě pro ty ostatní. Tyto dvě menší instance MAPF problému mohou výrazně zrychlit řešení než jejich spojení, díky tomu, že se jedná NP-těžký úkol[15].

Kapitola 4

Vlastní práce

V této kapitole se zaměřuji na samotné plánování autonomní dopravy, které jsem navrhl. Uvažuji o situaci, kde sami lidé neřídí automobil, ani nemohou jinak ovlivnit trasu, po které pojedou. Po ulicích se pohybují pouze sdílená autonomní vozidla, která jsou všechna stejného typu. Požadavky na přepravu vznikají od lidí, kteří chtějí využít vozidla k přesunu ze své pozice na jiné místo. Kromě osob se automobily mohou starají také o přepravu všech možných věcí po městě.

Aby tento celý dopravní systém mohl fungovat, je potřeba efektivní centralizované řízení dopravy, které zaručí, že systém neselže. Příkladem selhání je kolize více automobilů či zablokování dopravy, kde někdo nemůže pokračovat v cestě do cíle. Dalším požadavkem je optimalizace plánování tras. Lze se zaměřit na celkovou ujetou vzdálenost všech aut, čímž dochází k celkové úspoře energie, nebo na nejdelšího dobu trvání úkolu.

4.1 Definice problému

Centralizované řízení autonomní dopravy je v reálném světě komplexní problém, který zahrnuje spoustu faktorů. Aby bylo možné toto plánování algoritmicky vyřešit, je zapotřebí vymezit si prvky, které jsou důležité, a které je možné zanedbat.

Nejdůležitějším faktorem jsou samotná vozidla. Jsou uvažována klasická vozidla obdélníkového tvaru s poměrem stran 7:3 a možností jet dopředu a zatáčet pod určitým úhlem. Příklad takového vozidla je na obrázku 4.1. Automobily mohou jezdit pouze po silnicích. V případě, že vozidlo nejezdí po silnici, musí stát na určeném parkovišti, které má danou kapacitu. Silnice má vždy 2 pruhy a dopravní prostředky jezdí na její pravé straně. Silnice může vést rovně, nebo zatáčet pod úhlem 90°.

Úkoly jsou definovány potřebou zákazníků o přepravu. Pro splnění úkolu musí nejprve zvolené vozidlo dojet na vyzvedávací pozici úkolu a poté na pozici doručovací. V moment, kdy vozidlo doručuje nějaký úkol, nemůže začít žádný

Obrázek 4.1: Koncept zcela autonomního vozidla od společnosti Bosch

jiný. Kromě obyčejných úkolů může vzniknout také urgentní požadavek, který je potřeba splnit s nejvyšší prioritou. Toto simuluje například nutný průjezd jednotek integrovaného záchranného systému.

Zanedbáni jsou chodci a prvky dopravního systému s nimi spojené – například silniční přechod.

Pohyb automobilů po silnici je spojitá činnost, ale na samotné plánování je možné pohlížet čistě diskrétně. Silnice lze jednoduše rozdělit na jednotlivé úseky, kde na jeden úsek je možné umístit právě jeden automobil do každého pruhu s bezpečnostními mezerami na obou stranách. Z celé mapy města vznikne tedy síť silničních úseků, mezi kterými se vozidla mohou pohybovat. Plánování je poté zjednodušeno na hledání tras mezi těmito silničními úseky.

4.1.1 Stavový prostor

Celý problém řízení dopravy lze převést na multi-agentní problém vyzvedávání a doručování, kde agenti představují vozidla. Prostředí plánování dopravy lze převést do stavového prostoru, který definuje jednotlivé dovolené stavy systému a přechody mezi nimi.

Formálně je stavový prostor je zde definován jako trojice (G, A, S), kde G je graf představující mapu silnic, A je množina všech agentů v systému a S množina všech možných stavů.

Stav systému je definován dílčími stavy všech agentů (vozidel). Stav každého agenta a_i se skládá z:

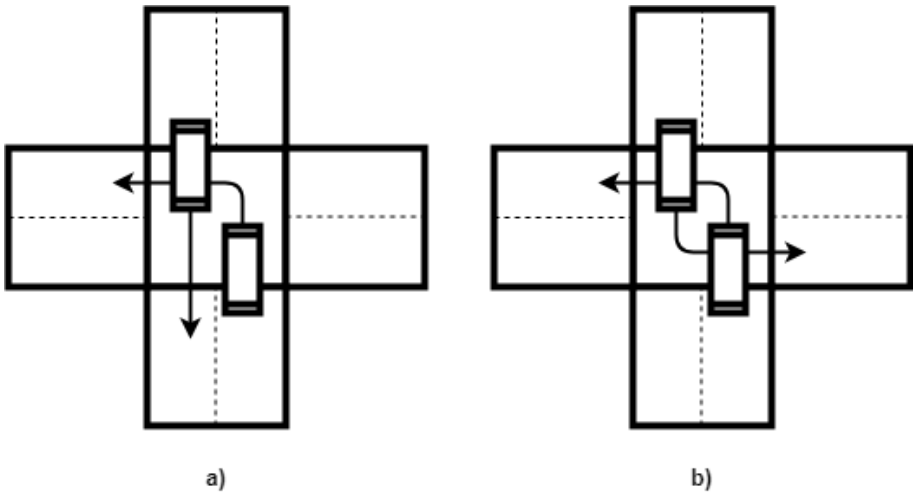
$(\mathbf{x}, \mathbf{y})$ – pozice úseku mapy, na kterém se agent nachází,

t – časový okamžik stavu (1, 2, 3, ...),

o – směrová orientace agenta (0, 1, 2, 3).

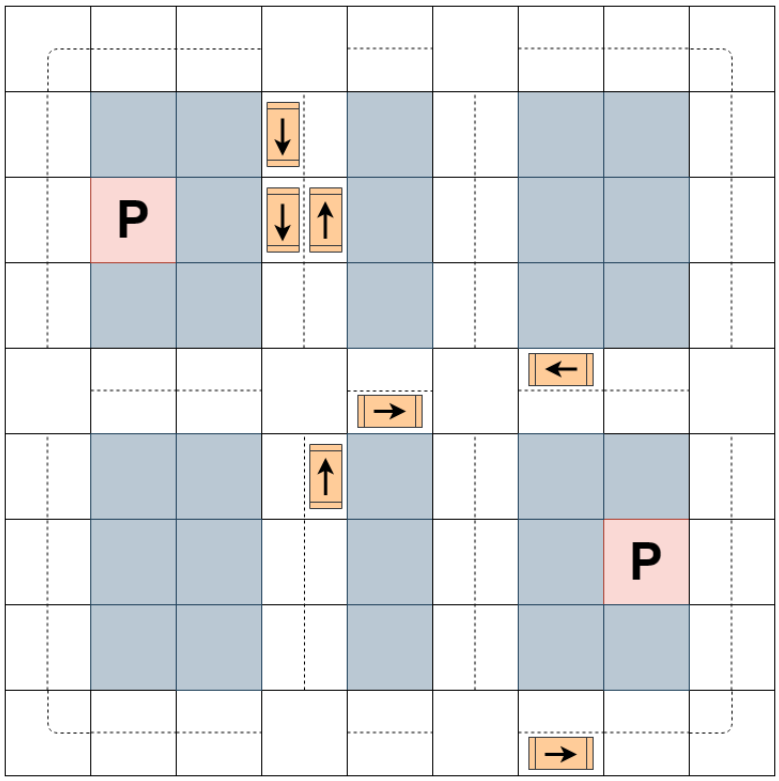
Přechod mezi stavy agenta je určen omezeními. S každým přechodem se zvyšuje čas, ve kterém se agent nachází. Je tedy možné přejít pouze do stavu, který má čas o jednu jednotku vyšší. Přechod mezi pozicemi agenta je omezen jak existencí silnice na dané pozici, tak i orientací v původním stavu. Je zakázán pohyb agenta v opačném směru jeho orientace, což by v reálném světe představovalo jeho couvání. Při zachování směru se orientace nemění, v případě se upraví o 1 v závislosti na novém směru.

Pohyb mezi stavy agenta je také omezen ostatními agenty. Stav systému je tedy validní, pokud stavy všech agentů mezi sebou nekolidují. Nepřípustné je, aby dva a více agentů byli ve stejném stavu. Pozice dvou agentů může být v jeden časový okamžik stejná, pouze pokud se nejedná o křižovatku silnic. Výjimkou je situace, kdy orientace dvou agentů je opačná, tedy míří proti sobě v jiných pruzích.



Obrázek 4.2: Příklady přechodu mezi stavy agentů

Přechod mezi stavy systému není dán pouze validitou stavů, ale také jsou kladeny podmínky na přechody mezi stavy agentů, aby nekolidovali mezi sebou. Na obrázku 4.2 a) je příklad, kdy původní i cílový stav je validní, ale přechod mezi nimi by způsobil kolizi. Na obrázku 4.2 b) nejde o zakázaný přechod, protože vozidla se v tomto případě mohou minout pravými boky, jak to dělají při běžném provozu.



Obrázek 4.3: Příklad stavu systému

Příklad stavu systému zobrazuje obrázek 4.3.

Počáteční stav systému neobsahuje žádné agenty. V každém časovém kroku může na pozicích označených jako parkoviště přibýt nový agent nebo může zaniknout.

4.1.2 Úkoly agentů

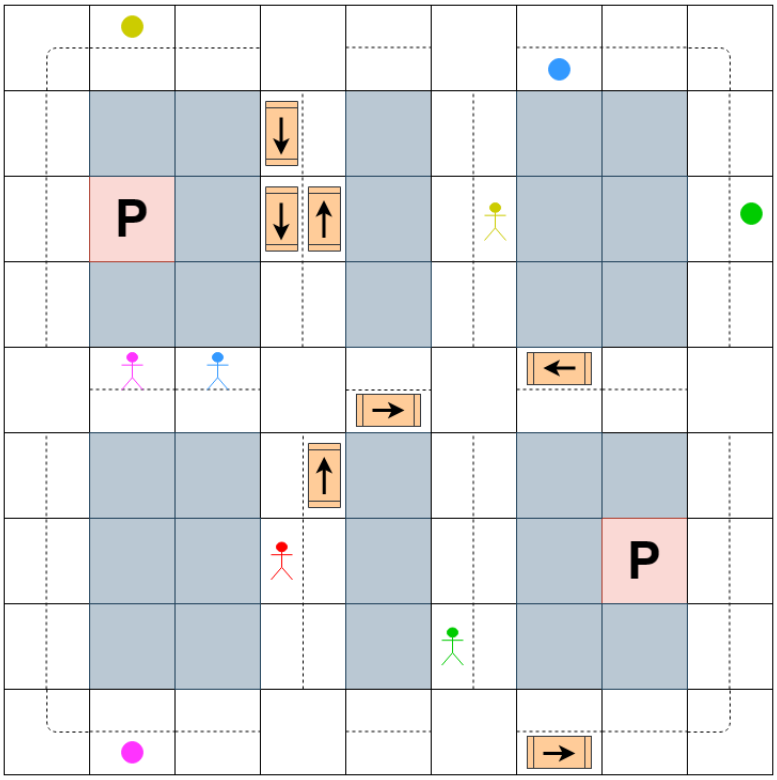
Úkoly agentů představují všechny přepravní požadavky dopravního systému, které jsou potřeba splnit. Každý úkol je definován vyzvedávací pozicí na mapě, na kterou musí agent dorazit, aby mu byl úkol přidělen, a doručovací pozicí, na kterou je poté potřeba se přemístit.

V reálném světě by automobil musel na každé z těchto pozic zastavit a zaparkovat pro vyložení/naložení zásilky a strávil by zde předem neurčitou dobu. Pro potřeby simulace je tento fakt ignorován. Pozice úkolu se může nacházet na jakémkoliv úseku silnice, avšak ne v místě křižovatky.

Úkoly vznikají postupně v čase a je potřeba je průběžně plnit. Doba od vzniku požadavku na úkol po dobu jeho splnění ukazuje, jak rychle daný úkol trval. Pro efektivní plánování řízení autonomní dopravy je kromě celkové

energetické náročnosti tedy také důležité, jak průměrně dlouho trvá úkoly plnit.

Příklad dopravního systému s úkoly je vidět na obrázku 4.4. Vyzvedávací pozice jsou zde vyznačeny panáčky a jejich odpovídající doručovací pozice je dle barvy znázorněna puntíkem.



Obrázek 4.4: Příklad stavu systému s úkoly

4.2 Použité algoritmy

U problému řízení autonomní dopravy je důležité správně vybrat algoritmus pro plánování tras, aby celý přepravní proces probíhal co nejplynuleji. Kritériem výběru takového algoritmu je minimální energetická náročnost systému, ale je zapotřebí se zamyslet také nad tím, jak by tento systém byl realizovatelný v reálném prostředí. Optimální plánovací algoritmy sice zaručují nejlepší výsledky, ale omezený dostupný výpočetní výkon by pro rostoucí počet agentů mohl být fatální.

Pro tuto práci jsem vybral celkem tři algoritmy, které jsou určeny pro multi-agentní vyzvedávání a doručování. Každý z nich přináší své výhody a nevýhody.

Token Passing(TP) – Tento algoritmus vyniká především nenáročností na výpočetní výkon, který se rozkládá mezi všechny agenty. Nevýhodou je, že primitivně pracuje s rozdělováním úkolů mezi agenty a výsledky plánování nemusí být optimální. Předpokládám, že tento algoritmus bude mít horší výsledky než další dva zvolené algoritmy.

Token Passing with Task Swaps(TPTS) – Vylepšení při rozdělování úkolů přináší tento algoritmus, kde je možné, aby si agenti úkoly vyměňovali, což si vybírá daň na výpočetním výkonu. Přesto se používá distribuovaný výpočet tras mezi agenty jako v předchozím bodě. Ačkoliv zde nedochází k optimálnímu plánování tras, výsledky tohoto algoritmu se mohou blížit k těm nejlepším.

Lifelong Centralized Pickup and Delivery(LCPD) – Nejlepší výsledky jsou očekávány u tohoto algoritmu, kde se trasy pro všechny agenty přepočítávají téměř v každém kroku a je zde zaručeno nalezení optimálního řešení. Kombinace častého plánování a hledání kolizí mezi trasami agentů má velký vliv na výpočetní výkon a pro velké množství agentů by to mohlo způsobovat nevyřešitelný problém.

Všechny tři algoritmy je zapotřebí modifikovat, aby je bylo možné na tomto problému použít. Hlavní změny se týkají prohledávání stavového prostoru pomocí algoritmu A*, který je používán v každém algoritmu. Další úpravy jsou zapotřebí z důvodu VIP požadavků či vhodného parkování.

4.3 Přidělování úkolů

Úkoly pro agenty vznikají dynamicky v čase a je zapotřebí se snažit je optimálně rozdělovat mezi agenty, aby žádný úkol nečekal na dokončení dlouho. Jde zde proti sobě minimalizace doby čekání úkolu a snaha plánovače minimalizovat energetickou náročnost systému.

Zavedl jsem proto heuristiku, která řeší, jaký úkol má pro konkretního agenta prioritu. Každý úkol má parametr, který reprezentuje čas, kdy vstoupil do systému, tedy jak dlouho čeká na začátek vyřizování. A každý agent může přesně nebo přibližně určit, jak daleko je od vyzvedávací pozice daného úkolu. Vzhledem k diskrétnímu plánování má zde doba čekání a vzdálenost podobný význam. Vztah každého agenta a nevyzvednutého úkolu je možné vyjádřit číslem, a to jako rozdíl mezi vzdáleností a dobou čekání úkolu.

Vzhledem k rozdílnosti algoritmů se vzdálenost agenta k úkolu může vypočítávat jinak.

TP – Zde probíhá jednostranné vybírání úkolů ze strany agenta, který právě provádí plánování. Budoucí omezení v čase a prostoru jsou předem známá a pro zpřesnění výpočtu vzdáleností je možné využít algoritmus

A*, stejně jako kdyby si agent daný úkol již vybral. Touto možností je vybrán zaručeně nejbližší úkol a trasu pak již není nutné plánovat. Po přidělení úkolu agentovi není úkol pro jiného agenta k dispozici při výběru.

TPTS – Podobně jako v předchozím bodě se jeden agent rozhoduje mezi všemi úkoly. Avšak jsou uvažovány i již přidělené, ale nevyzvednuté úkoly. Vzdálenost je možné zjistit přesně. Pokud je ale k vybranému úkolu přidělen jiný agent a vzdálenost plánovacího agenta je menší než původního majitele, je zapotřebí zajistit, že je mu přidělen nový úkol nebo má ve své situaci možnost naplánovat trasu k nejbližšímu parkovišti. Pokud toto není zaručeno, není možné úkoly vyměnit.

LCPD – U tohoto algoritmu nastává problém, protože ve fázi přidělování úkolu nejsou známa žádná budoucí omezení a délka trasy nalezená pomocí A* by ukazovala pouze ideální případ. Proto jsem zde zjednodušil hledání vzdáleností mezi agenty a úkoly na euklidovskou vzdálenost bodů na mapě, což v tomto případě dává dobrý odhad.

Ve všech algoritmech si tedy agenti udržují přehled o svém přiděleném úkolu, který se u TPTS a LCPD může měnit. Ve chvíli, kdy agent dorazí na vyzvedávací pozici tohoto úkolu, začíná doručování a nemůže mu být další úkol přidělen ani odebrán.

Kromě již aktivních agentů je při rozdělování úkolů také zapotřebí uvažovat o neaktivních na parkovištích. Pokud je na daném parkovišti alespoň jeden čekající agent, je tato pozice zahrnuta do rozdělování. U LCPD má rovnou pozici mezi všemi aktivními agenty ve fázi rozdělování před plánováním. U TP a TPTS jsou tito agenti uvažováni až po plánování v případě, že existuje více úkolů než aktivních agentů.

Algoritmus 5: TP - Task assignment

```
1  Function TP_task_assignment(a, τ):
2      best_score ← NULL
3      t' ← NULL
4      foreach t in τ do
5          r_t ← A*(a, t)
6          score ← len(r_t) − t.in_time
7          if score < best_score then
8              best_score ← score
9              t' ← t
10     Assign(a, t')
11     τ.remove(t')
```

Algoritmus 6: TPTS - Task assignment

```
1  Function TPTS_task_assignment(a, τ):
2      options ← ∅
3      foreach t in τ do
4          r_t ← A*(a, t)
5          score ← len(r_t) − t.in_time
6          options.add((t', score))
7      Sort options by score
8      foreach o in options do
9          a' ← Get_Owner(o.task)
10         if a' = NULL then
11             Assign(a, o.task)
12             return
13         if a'.score < o.score then
14             continue
15         if PlanRoute(a')issuccess then
16             Assign(a, t')
17             return
```

Algoritmus 7: LCPD - Task assignment

```
1  Function LCPD_task_assignment(A, τ):
2      distances ←
3      foreach t in τ do
4          foreach a in A do
5              d ← Euclid(t, a)
6              score ← d − t.in_time
7              distances[t].add((a, score))
8          Sort distances[t] by score
9      while True do
10         if All tasks assigned then
11             return
12         foreach free a in A do
13             foreach t in distances do
14                 if distances[t].first = a then
15                     Assign(a, t)
16                     Remove a from distances
```

4.4 Plánování tras

Plánování tras je nejdůležitější součástí řízení dopravy. V tomto kroku mají všichni agenti přidělený úkol nebo jsou volní a plánují cestu na parkoviště.

Potřebují se proto dostat ze své pozice na cílovou pozici. Existují tedy 3 možné cíle agenta:

Vyzvedávací pozice – Agent dostal přidělený úkol a potřebuje se přemístit na vyzvedávací pozici.

Doručovací pozice – Agent právě není volný, plní svůj úkol a míří na doručovací pozici.

Parkoviště – Žádný úkol nebyl agentovi přidělen nebo mu byl odebrán, a proto směřuje na nejbližší parkoviště.

Při plánování je cílem minimalizovat celkovou sumu délek všech tras. I čekání na jedné pozici je bráno jako prodloužení trasy, protože se prodlužuje doba splnění úkolu a obsazenost agenta.

Frekvenci opětovného plánování tras je třeba zredukovat tak, aby k němu nedocházelo ve zbytečných případech. Algoritmy TP a TPTS není možné v tomto smyslu optimalizovat, protože plánování již probíhá opravdu jen v případech, kdy to agenti sami potřebují. Na rozdíl od nich algoritmus LCPD se v každý časový krok snaží najít nové optimální trasy. Kombinace tohoto častého plánování a obecné výpočetní náročnosti algoritmu CBS, na kterém je LCPD založen, může vzniknout problém, že plánování nebude dokončeno včas a celý dopravní systém by musel být zastaven, než budou známy nové trasy.

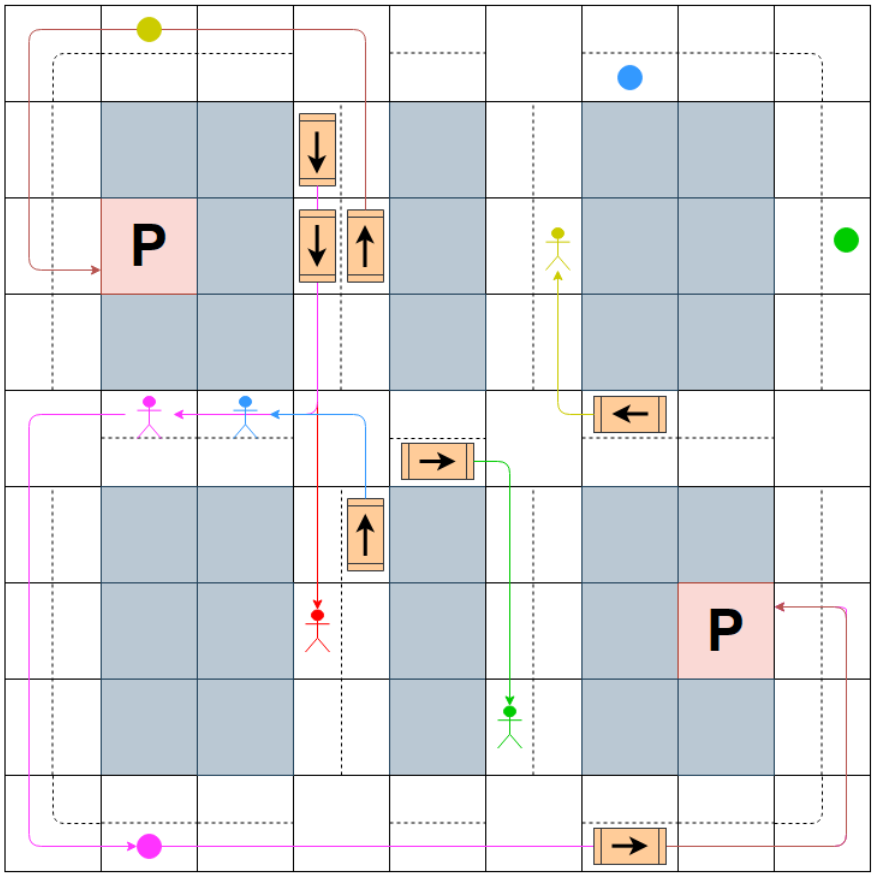
Optimalizace spočívá v tom, že je není nutné přeplánovávat v každém kroku, ale pouze tehdy, když se v systému stane určitá akce. Nutnost znovu naplánovat trasy nastává u algoritmu LCPD po těchto akcích:

- vznikne nový úkol,
- agent dokončí svůj úkol.

Příklad systému s přidělenými úkoly a naplánovanými trasami jde vidět na obrázku 4.5. Dvě vozidla nemají přidělený úkol a míří na parkoviště. U růžového úkolu je vidět, jak vypadá celá trasa vozidla ze současné pozice na vyzvedávací, dále na doručovací pozici a nakonec na nejbližší parkoviště. Takto by trasa byla naplánována u algoritmů TP a TPTS. Ostatní vozidla mají naplánované trasy pouze na vyzvedávací pozici, jak je tomu u algoritmu LCPD.

4.4.1 Nebezpečí deadlocku

Při plánování je nutné se vyvarovat situaci, kdy pro agenta nebude možné neplánovat žádnou trasu k úkolu nebo na parkoviště. Takto stojící agent by blokoval trasy ostatních agentů a celý systém by se mohl dostat do stavu, kdy tuto situaci nelze vyřešit.

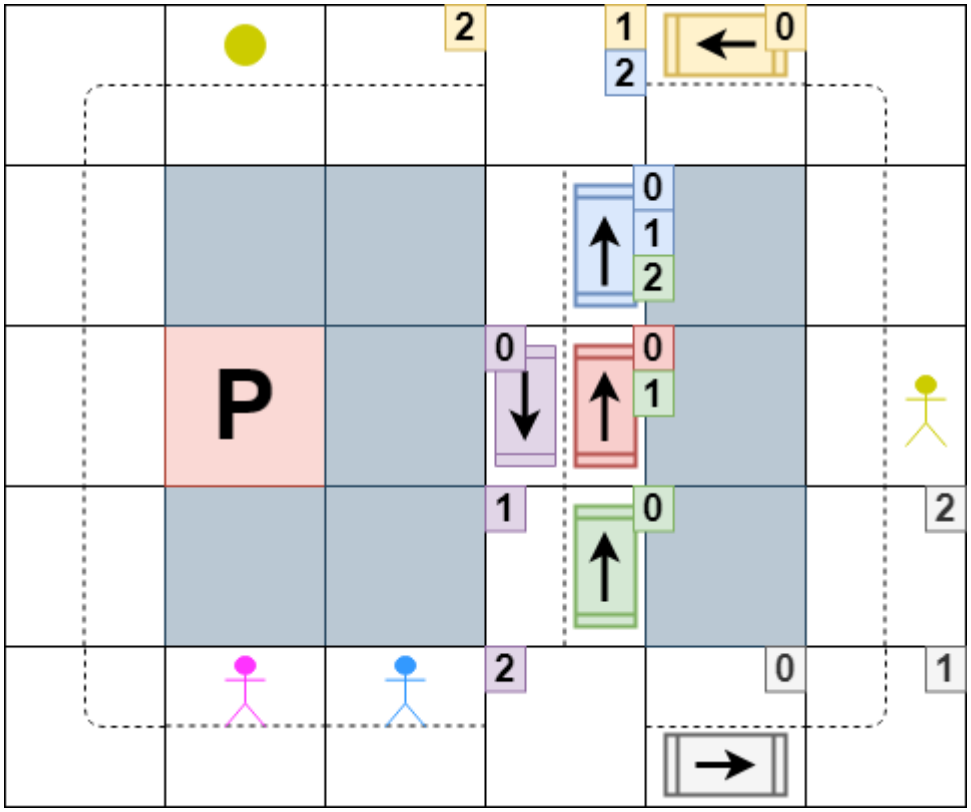


Obrázek 4.5: Stav systému s úkoly a naplánovanými trasami

Na obrázku 4.6 je příklad situace, kdy došlo ke špatnému plánování a agent nemůže naplánovat další trasu. Toto konkrétně se může stát u algoritmu TP nebo TPTS, kdy si agenti plánují trasy individuálně dopředu, ale po dokončení trasy s nimi není dále počítáno. Zde červený agent v kroku 0 dokončil doručování svého úkolu, ale ostatní agenti si zarezervovali trasy tak, že červený se v dalším kroku nemůže pohnout a ani nemůže zůstat na svém místě. Pro vyřešení této situace by všichni dotčení agenti museli přeplánovat své trasy a došlo by k celkovému zpoždění. Nepřímo by mohl být ovlivněn celý systém, protože jiní agenti počítali s původními rezervacemi tras a přizpůsobili tomu plánování.

Z těchto důvodů je zapotřebí plánovat trasy agentů tak, aby v jakékoliv situaci nezůstali stát na silnici bez možnosti dalšího pohybu. V tomto teoretickém prostoru je za bezpečné místo považováno pouze parkoviště. Každý algoritmus přistupuje k tomuto problému jinak:

TP – Agenti vyjíždějí z parkoviště až při přidělení úkolu a zarezervování si své budoucí trasy. Při plánování uvažují o trase z parkoviště na vyzvedávací



Obrázek 4.6: Ukázka deadlocku při plánování

pozici, následně na doručovací a nakonec zase na nejbližší parkoviště. Pokud tyto menší trasy nejsou nalezeny, plánování je neúspěšné a agent zůstává na parkovišti. Aby nemuselo docházet k vracení se na parkoviště, je po doručení úkolu testováno přidělení nového úkolu a nalezení trasy. Pokud je možné takovou trasu naplánovat, je agentovi zrušena původní rezervace cesty na parkoviště a zarezervována nová trasa.

TPTS – Plánování u tohoto algoritmu probíhá podobně jako u jeho jednodušší verze, avšak zde nastává problém s výměnnou úkolů. Agent již může být na cestě z parkovací pozici, ale jiný agent mu úkol sebere. V těchto případech je nutné kontrolovat, zda agent, který přijde o úkol, může úspěšně naplánovat trasu k jinému úkolu nebo na parkoviště. Pokud to možné není, výměna nemůže být provedena.

LCPD – Díky přeplánovování všech tras najednou je u tohoto algoritmu deadlock vyloučen. Trasy pro agenty jsou zde plánovány ze současné pozice agenta do vyzvedávací pozice a následně na doručovací.

4.4.2 Úprava řídících algoritmů

4.4.2.1 TP

Řízení agentů pomocí algoritmu Token Passing[19] v mé práci funguje takto. V každém časovém kroku pro plánování je prováděna tato série aktivit.

1. **Parkování** – Pokud je nějaký agent na pozici jakéhokoliv parkoviště a nemá přidělený žádný úkol, je považován za přebytečný a je zaparkován. Tím pádem agent zmizí ze systému.

2. **Plánování trasy** – Všichni agenti jsou postupně kontrolováni a pokud některý nemá přidělený úkol, tak je snaha mu jej přidělit. Jsou uvažovány všechny volné úkoly. Pomocí algoritmu A* je nalezen ten s nejbližší vyzvedávací pozicí a následně je doplánována trasa na doručovací pozici na nejbližší parkoviště vzhledem k pozici, kde agent bude po dokončení úkolu. Pokud takovou trasu není možné najít pro žádný úkol, agent pokračuje ve své naplánované cestě na parkoviště. V opačném případě jsou budoucí plány vymazány ze sdílené rezervační mapy a je zadána nová trasa.

3. **Přidání nového agenta** – Jsou uvažována všechna parkovací místa, kde je alespoň jeden agent. Jestli je možné naplánovat trasu jako v předchozím bodě, tak je agent přidán do systému na pole parkoviště.

4. **Pohyb agentů** – Na základě naplánovaných tras a současného času je všem agentům rozeslán příkaz, kterou činnost mají uskutečnit.

5. **Kontrola úkolů** – Všechny úkoly jsou kontrolovány, zda nedošlo ke změně jejich stavu tím, že agent dorazil na jejich pozici.

4.4.2.2 TPTS

Fungování algoritmu Token Passing with Tasks Swaps[19] je podobné jako u TP a liší se především v bodě 2. Plánování trasy. Plánování je vždy spuštěno pro konkrétního agenta. Nejprve jsou vypočítány trasy a vzdálenosti ke všem úkolům, které ještě žádný agent nezapočal. Tyto výsledky jsou poté seřazeny podle vzdáleností a postupně procházeny. Pro každou možnost se kontroluje, zda je úkol přidělen jinému agentovi. Pokud ano, musí být vypočítána také jeho vzdálenost k danému úkolu a výsledky jsou porovnány. Jestli má dojít k výměně úkolu, musí být úspěšně naplánována budoucnost druhého agenta.

4.4.2.3 LCPD

Řízení agentů pomocí algoritmu LCPD[19] funguje na jiné bázi než u TP a TPTS. Plánování se tu provádí vždy pro všechny agenty najednou a pouze pokud došlo ke změně v systému. Každý krok řízení probíhá takto:

1. **Parkování/Příprava agentů** – Agenti bez úkolu na parkovišti jsou odebráni ze systému. Zároveň na všech parkovištích, kde je to možné, se jeden agent připojí do systému, aby se mohl zúčastnit plánování v dalším kroku.

2. **Plánování tras** – Každému volnému agentovi je přiřazen volný úkol nebo cílové parkoviště. Na základě toho každý agent zná svou plánovanou pozici. Pomocí CBS jsou trasy nalezeny.

3. **Parkování aut** – Pokud nějakému agentovi stojícímu na parkovišti nebyl přidělen úkol, je opět odstraněn ze systému.

4. **Pohyb agentů** – Na základě naplánovaných tras a současného času je všem agentům rozeslán příkaz, kterou činnost mají uskutečnit.

5. **Kontrola úkolů** – Všechny úkoly jsou kontrolovány, zda nedošlo ke změně jejich stavu tím, že agent dorazil na jejich pozici.

Do multi-agentního hledání tras pomocí CBS vstupuje mapa rozdělená na plánovací úseky, současná pozice, orientace agentů a jejich cílová pozice. Pomocí upraveného algoritmu A* jsou nalezeny trasy pro všechny agenty bez ohledu na ostatní. U každého řešení se jeho cena počítá jako suma délek všech tras včetně čekání. Řešení, které nemá pro libovolného agenta naplánovanou trasu, má neurčenou cenu. Jednotlivá řešení se neprochází stromově podle jejich nalezení, ale přednost má to s nejnižší cenou. Každé řešení se validuje a hledají se dva druhy konfliktů:

uzlové – Pokud jsou dva agenti ve stejný čas na stejné pozici, vzniká uzlový konflikt. Výjimkou jsou situace, kdy je jejich orientace odlišná a zároveň nestojí v daný čas na křižovatce. Stát ve stejném směru na křižovatce je dovoleno.

hranové – Není dovoleno, aby se více agentů pohybovalo ve stejný čas, stejným směrem po stejné hraně mezi dvěma uzly.

Také je zakázáno pro každou dvojici agentů, aby z pozice, kdy stojí naproti sobě na křižovatce, jeden zabočil doleva a druhý jel rovně. Tento příklad je zobrazen na obrázku 4.2 a).

Nejprve jsou kontrolovány uzlové a poté hranové konflikty. Pokud je konflikt nalezen, vytvoří se nová řešení, přidají se omezující podmínky a vypočítá se cena. V případě, že je řešení validní, je vráceno do plánovače.

4.4.3 A* pro hledání tras

Upravený algoritmus A* se používá ve všech 3 plánovacích algoritmech pro nalezení trasy s omezeními. Oproti klasické implementaci je důležité uvažovat, že se agent s každým krokem posouvá také v čase.

Agent se ve svém stavovém prostoru může pohybovat podle své směrové orientace. Uvažovány jsou v každém kroku tyto 4 akce:

- pohyb rovně,
- zabočení doleva,
- zabočení doprava a
- čekání na místě.

Podle vykonané akce se musí pro příští krok také upravit orientace. Pohyb je také omezen silnicemi a je zakázáno, aby agent zůstával čekat na křižovatkách.

Dle plánovacího algoritmu jsou kontrolována také omezení:

TP, TPTS – Rezervační mapa zde udává polohu v čase všech agentů. Nový agent může vstoupit na pozici v daném čase, pokud je prázdná nebo se nejedná o křižovatku a druhý agent nemá stejnou orientaci.

LCPD – Zde jsou kontrolována současná uzlová a hranová omezení.

Přidání možnosti čekání na místě může způsobit zejména u hledání cest pomocí CBS problém. Při nemožnosti najít trasu se do fronty přidávají stále uzly se stejnou pozicí a čekací akcí. Je vhodné přidat omezující podmínku, která uzly s velmi vysokou g hodnotou již nepřidá do fronty.

4.5 Vylepšení řízení dopravy

4.5.1 VIP úkoly

Do této doby se při plánování uvažovalo pouze o úkolech, které mají prioritu pouze podle jejich stáří v systému. VIP úkoly simulují urgentní požadavek na dopravní systém, který je potřeba přednostně splnit před obyčejnými úkoly. Kromě co nejrychlejšího přidělení určitému agentovi, je vyžadováno, aby se tento agent dostal na vyzvedávací pozici v co nejmenším počtu kroků a poté co nejrychleji úkol doručil. Při plánování může dojít k situaci, kdy v systému je více VIP úkolů. V tomto případě mají mezi sebou prioritu jako obyčejné úkoly.

Plánování je zde obtížnější, protože je zapotřebí, aby v jakémkoliv stavu systému bylo možné VIP úkol splnit přednostně. Konkrétně úprava doposud naplánovaných tras musí být validní, aby po naplánování VIP trasy byl plánovač schopen najít cesty pro zasažené agenty.

Dle typu plánovacího algoritmu jsem navrhl tyto postupy.

TP, TPTS – Algoritmy TP a TPTS mají shodné plánování, které spoléhá na jistou budoucnost. V případě, kdy tuto budoucnost je nutné měnit nastává problém. Aby bylo možné pro VIP úkol naplánovat optimální trasu, může docházet k tomu, že bude nutné zrušit rezervace stávajících tras.

Možné extrémní řešení by bylo, že pokud se objeví nový VIP úkol, tak by došlo ke zrušení všech rezervací, následně k naplánování trasy pro daný úkol a na závěr k opětovnému rozposlání tokenu, aby si každý agent našel novou trasu. Pořadí, v jakém posílat token mezi agenty, by určovalo, jestli bude možné pro zbývající agenty znovu naplánovat trasu. V případě neúspěchu by bylo možné vyzkoušet jinou permutaci, ale nebyl by ani tak zaručen úspěch.

Rozhodl jsem se proto, že pro VIP úkol bude naplánována trasa, kde omezení budou klást pouze ostatní agenti se stejně důležitým úkolem. Následně bude zjištěno, s kterými agenty nastává konflikt. Trasa pro VIP úkol bude zarezervována a pouze konfliktním agentům bude naplánována nová trasu k jejich cíli. Pokud to, byť pro jednoho agenta, není možné, plánování VIP úkolu se odloží do dalšího kroku.

LCPD – U tohoto algoritmu, kde plánování všech tras probíhá vždy od začátku, je situace snazší. Jednou z možností, jak priorizovat VIP úkoly je, že se nejprve pomocí CBS naleznou trasy jen pro VIP úkoly a až poté se hledají cesty pro obyčejné úkoly, tak aby nebyly konfliktní s již naplánovanými.

Nakonec jsem zvolil jiný přístup. Využil jsem faktu, že CBS v iniciační fázi naplánuje trasy pro všechny agenty bez ohledu na ostatní. Díky tomu má agent s VIP úkolem vytvořenou zaručeně nejlepší cestu. Tyto trasy by však mohly být upraveny při hledání optimálního řešení ve prospěch obyčejného úkolu. Proto pokud je nalezen konflikt v řešení a každý z agentů má jinou prioritu, tak pro agenta s VIP úkolem nevznikne nové omezení a tato větev je ignorována. V případě konfliktu agentů se stejnou prioritou je situace řešena standardní cestou.

4.5.2 Parkování dle výskytu úkolů

Zaparkovat nepotřebné vozidlo na nejbližší parkoviště má tu výhodu, že nedochází ke zbytečnému cestování vozidla a celková energie je šetřena. V teoretickém prostředí můžeme uvažovat, že úkoly vznikají náhodně na různých částech mapy. V reálném prostředí se to ovšem neděje. Existují oblasti, kde je větší či menší pravděpodobnost, že je tam zapotřebí vyzvednout či doručit nějaký úkol. Například v centru města bude v určitou denní dobu poptávka po doručení do této oblasti a odpoledne naopak po vyzvednutí. Pokud vozidlo zaparkuje na nejbližším parkovišti, avšak v oblasti, kde příliš vyzvedávacích

pozic nebývá, prodlužuje dobu, po kterou pojede do jiné oblasti, kde mohl zaparkovat dříve.

Jedním z možných vylepšení řízení dopravy je tedy vnímat, kde vznikají nové požadavky a na základě toho se snažit vozidla zaparkovat na nejvhodnější pozici. Stále je ovšem nutné přemýšlet nad efektivitou pohybu vozidel. Proto by nikdy nebude rozumné vozidlo posílat přes celé město s myšlenkou, že tam spíše bude dříve nový požadavek.

Pro rozhodování, na které parkoviště vyslat vozidlo, je stále důležitá jejich vzdálenost. Je ale nutné zvýhodnit parkoviště, okolo nichž vzniká více úkolů. Každé parkoviště má svoji váhu, která udává, jak je pro systém důležité. Při vzniku nového úkolu je jeho vyzvedávací pozice zanesena do rozhodovacího systému a váhy parkovišť upraveny. Pro novou pozici x a sadu parkovišť $p_1, ..., p_i$ s váhami $w_1, ..., w_i$ a konstantami k a u dojde k těmto úpravám.

1. Pro každé parkoviště p_j je spočítána euklidovská vzdálenost e_j k nové pozici x.
2. Přírůstek váhy n_j je spočítán jako $n_j = e_j^{-u} * k$.
3. K váze w_j je přičten přírůstek n_j.

Váha každého parkoviště je změněna, avšak váha bližších parkovišť více než vzdálenějších. Při rozhodování, kam zaparkovat vozidlo vítězí parkoviště jehož výsledek výrazu w_j/e_j po dosazení je nejvyšší. Konstanty k a u je nutné optimálně nastavit dle prostředí.

4.6 Simulace

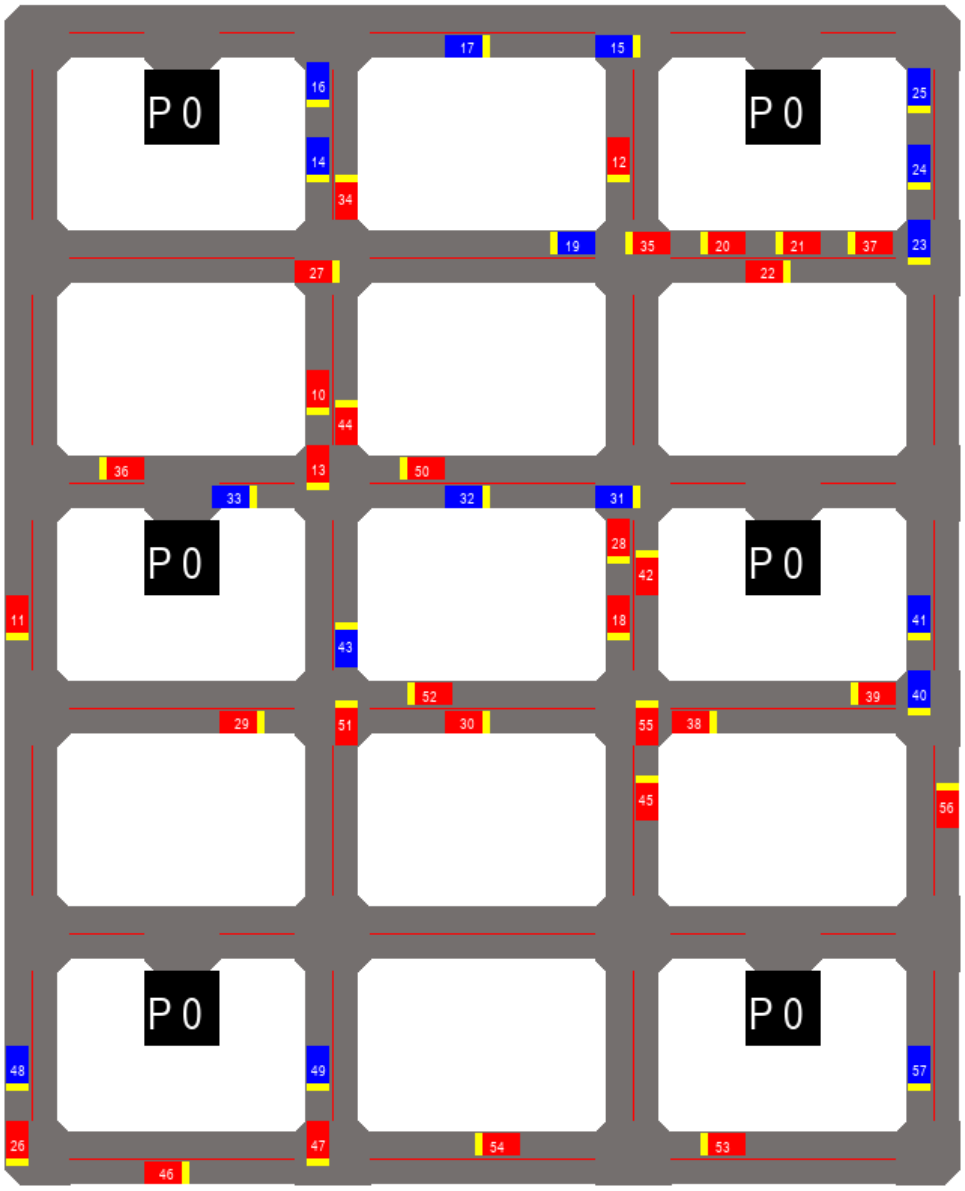
Simulace slouží k otestování navržených algoritmů a jejich vyhodnocení. Prostředí je reprezentováno jako 2D mřížky, kde je možné vymodelovat mapu silnic. Každý bod mřížky je čtverec o velikosti vozidla, stejně velký jako úsek silnice při plánování.

Celková velikost mapy je definovaná její šířkou a délkou, tedy počtem čtverců. Vizuálně může být silnice na jakémkoliv dílku mřížky. Podle okolních dílků vznikne ve finále jeden z následujících útvarů:

- rovná silnice,
- zatáčka,
- křižovatka ve tvaru T,
- křižovatka ve tvaru +,
- slepá ulice nebo
- zástavba.

Na určených bodech jsou umístěna parkoviště, která slouží k parkování nepotřebných vozidel. Aby vozidlo mohlo vyjet z parkoviště, musí být na sousedním dílu silnice. Počet vozidel, která jsou na parkovišti, je určen počátečním stavem a nově zaparkovanými auty.

Ukázka vizuální stránky simulace je vidět na obrázku 4.7.



Obrázek 4.7: Vizuální stránka simulace s 48 vozidly a mřížkou 13x16

Úkol je dvojice vyzvedávací a doručovací pozice. Existují dva způsoby, jak nové úkoly vznikají:

dynamicky – Je určen maximální počet úkolů například vůči počtu vozidel a pokud je úkol splněn je nový náhodně vygenerován na nějakých pozicích na mapě.

staticky – Seznam úkolů je určen na začátku simulace formou vstupního souboru. Kromě pozic je nutné určit čas, kdy úkol vstoupí do systému.

Před začátkem simulace je potřeba určit podobu mapy, způsob vzniku nových úkolů, polohu parkovišť a plánovací algoritmus. Po spuštění není v systému žádné vozidlo a všechny čekají na parkovišti. Vozidla se pohybují plynule a v určitých intervalech se volá plánovač, aby vozidlům zadal jejich budoucí akce. Aby vozidlo mohlo zareagovat například na zatáčku je nutné, aby trasy byly naplánované ještě před tím, než se vozidlo dostane do středu plánovacího úseku.

Díky simulaci bude možné otestovat navržené řídící algoritmy. Dle zvoleného scénáře je potřeba upravit podmínku pro ukončení simulace. Při běhu simulace jsou postupně zaznamenávány splněné úkoly. Ty jsou uloženy do zadaného výstupního souboru. O každém úkolu se zaznamenávají tyto informace:

- ID úkolu – pořadí přidání do systému,
- vyzvedávací pozice,
- doručovací pozice,
- čas vstupu do systému,
- čas vyzvednutí,
- čas doručení,
- ID doručovacího agenta a
- urgentnost úkolu.

Při běhu simulace slouží k zobrazení aktuálního stavu systému, aby bylo možné vyhodnotit, jestli vznikají kolony či jak je provoz plynulý.

Kapitola 5

Testování

Předmětem testování je porovnání tří řídících algoritmů. Srovnání bude probíhat na základě jejich efektivnosti při řešení požadavků dopravního systému, ale také bude záležet na nárocích na výpočetní výkon.

V první řadě bude simulací testována schopnost plnit základní účel řízení dopravy, a to co nejrychleji zvládat vyřizování všech úkolů. Poté vyzkouším, jak si dané algoritmy poradí s různým počtem úkolů v jeden moment v závislosti na počtu dostupných vozidel. Kromě efektivního plánování tras pro vozidla je také důležité, jak jsou postupně plněny jednotlivé úkoly. Pokud by se jednalo o přepravu zákazníků, tak je důležitá doba mezi vstupem požadavku na přepravu do systému a vyzvednutím osob, aby nedocházelo k dlouhému čekání.

Pokud má řídící algoritmus k dispozici určitý počet vozidel, předpokládá se, že je dokáže všechny rovnoměrně využívat a nestává se, že jedno vozidlo v průměru vyřídilo mnohem více úkolů než ostatní vozidla.

Dalším bodem k testování je schopnost reagovat na různý počet VIP požadavků a zároveň udržet schopnost uspokojovat i obyčejné úkoly.

Poslední částí testování bude měření nároků na výpočetní výkon, který je potřebný pro plánování tras. S rostoucím počtem vozidel by dopravní systém měl být stále schopen řídit dopravu.

Testování probíhá na mapě, která je na obrázku 4.7. Je to mřížka o velikosti 13x16 bloků s 20 křižovatkami.

5.1 Obecná výkonnost dle počtu agentů

Hlavním ukazatelem, jak efektivně algoritmy dokáží řídit dopravu, je schopnost, v co nejkratší době plnit úkoly v systému. Na dané mapě silnic poté záleží, kolik vozidel bude k dispozici ku celkové velikosti mapy. Menší počet vozidel představuje snazší plánování, ale je zde velmi důležité správné přidělování úkolů správným agentům. S rostoucím počtem vozidel je přidělování méně závažné, protože vozidla nejspíše budou rovnoměrně rozmístěna po mapě.

Avšak plánování tras začíná být obtížné, aby se minimalizovala celková energetická náročnost systému.

Pro měření výkonnosti jsem navrhl scénář simulace tak, že je určen počet dostupných vozidel a ty budou po určitou dobu plnit postupně zadávané úkoly. Cílovým ukazatelem je počet splněných úkolů. Jelikož každý algoritmus má jinou efektivitu plánování a přidělování, rozhodl jsem se, že úkoly nebudou zadávány pro všechny algoritmy stejně. Pokud by efektivnější algoritmus plnil úkoly rychleji než jiný, tak by stále musel čekat na nový úkol. Z toho důvodu se úkoly pro každý algoritmus generují náhodně tak, aby v každý moment byl počet nedokončených úkolů stejný jako počet dostupných vozidel. Testování bude probíhat postupně pro 10, 20 a 30 dostupných agentů.

Obrázek 5.1: Počet splněných úkolů dle počtu agentů

Výsledek simulace je vidět na grafu na obrázku 5.1.

Pro nejnižší počet vozidel nelze pozorovat výrazné změny mezi algoritmy TPTS a LCPD. Na konci simulace vede LCPD s počtem 259 splněných úkolů, avšak TPTS má pouze o 2,3 % méně. Tento podobný výsledek připisuji právě snadnému plánování tras pro malý počet agentů, kde rozdíl mezi optimální variantou a individuálními rezervacemi se velmi výrazně neprojeví. Avšak kvůli špatnému přidělání úkolů mezi agenty, TP zaostává o 19 % oproti LCPD.

Pro 20 dostupných agentů má opět nejvíce splněných úkolů LCPD, a to 559. Rozdíl mezi TPTS se zvýšil na 4,5 % a TP se propadlo o 24 %.

Pro nejvyšší počet testovaných agentů se potvrdil rostoucí trend zvětšování

rozdílů mezi zkoumanými algoritmy. Pro 30 agentů řídící algoritmus LCPD zvládl splnil 870 úkolů. TPTS zde zaostává již o 6 % a TP o 31,5 %. Nutno zmínit, že již pro 30 agentů každý algoritmus v průměru odbavil více než 1 požadavek za jednotku času.

Tento procentuální rozdíl mezi algoritmy by se pro vyšší počet dostupných agentů postupně zvyšoval.

5.2 Závislost počtu agentů na počtu úkolů

Celý dopravní systém je proměnné prostředí, kde počet aktuálních požadavků na přepravu závisí na mnoha faktorech, nejvíce na denní době. Proto by si řídící algoritmus měl umět efektivně poradit se situacemi, kdy je v jednu chvíli málo úkolů, nebo naopak mnoho. Při menším počtu je důležité, aby byla zachována nízká energetická náročnost systému, a proto není účelné využívat více vozidel, než je opravdu potřeba. Při velkém množství úkolů více rozhoduje spokojenost zákazníků, kterou lze měřit například průměrnou dobou mezi vstupem do systému a doručením.

Abych tyto situace mohl simulovat, vytvořil jsem scénář, kdy na mapě jezdí 20 vozidel, a měnil poměr mezi počtem agentů a úkolů v každém čase. Nejprve je testován poloviční počet úkolů než vozidel, tedy 10. Následně je počet úkolů stejný jako počet vozidel a nakonec je v každou chvíli v systému 40 úkolů, což odpovídá dvojnásobku počtu vozidel.

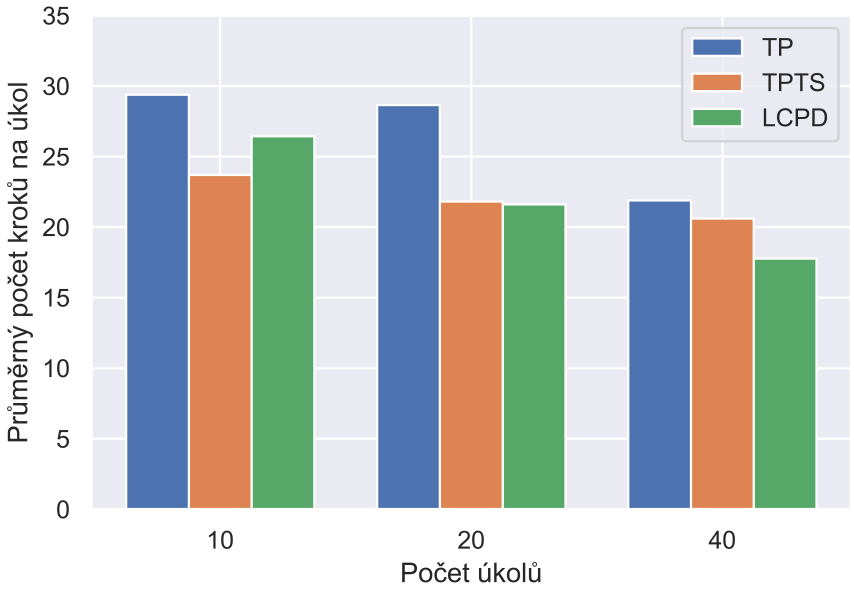
Uživatelská spokojenost je měřena jako průměrná doba od vstupu úkolu do systému po jeho splnění a efektivita přidělování a plánování je vyjadřována jako průměrný počet pohybů vozidla na splnění úkolu.

Na obrázku 5.2 je graf zobrazující výsledky průměrné krokové náročnosti na jeden úkol. Průměrnou dobu, kterou úkol stráví v systému, ukazuje graf na obrázku 5.3.

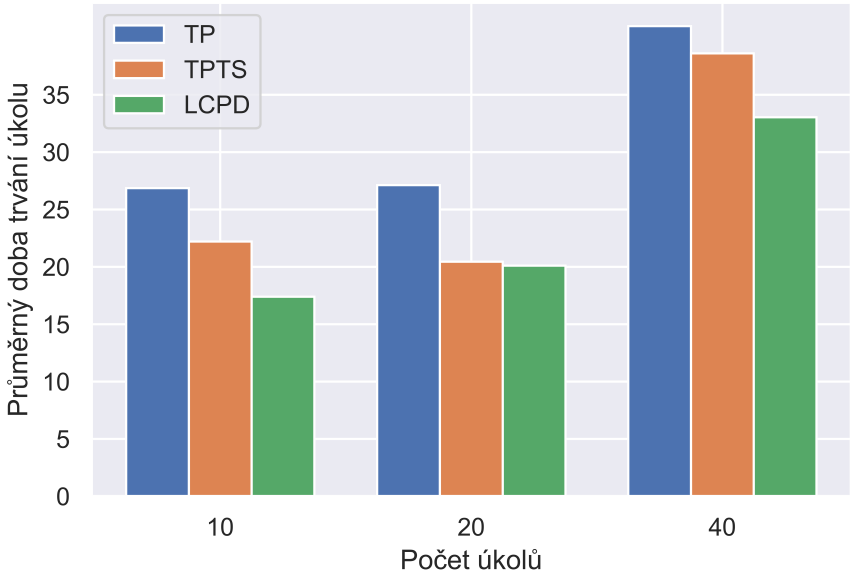
Pro poloviční počet úkolů, než je vozidel v systému, dosahuje algoritmus LCPD nejnižší průměrné doby trvání úkolu, která je o 22 % nižší než v případě TPTS a o 36 % nižší než u TP. V průměrném počtu kroků na jeden úkol sice vítězí TPTS, avšak LCPD splnilo o 27 % více úkolů. Z toho vyplývá, že TPTS se spíše snažilo využívat v jeden moment méně vozidel pro úsporu energie, což mělo negativní vliv na průměrnou době trvání úkolu.

Ve scénáři, kdy bylo v každý moment stejný počet vozidel jako úkolů, dopadly výsledky pro TPTS a LCPD velmi podobně. Vozidla byla celou dobu vytížena a nezajížděla na parkoviště, tudíž součet všech kroků, které vozidla provedla, byl pro všechny tři algoritmy stejný. Průměrný počet kroků proto závisel na počtu dokončených úkolů, kde LCPD jich splnilo o 5 více než TPTS. Podobně je tomu i u průměrné doby úkolu v systému, kde jsou výsledky pro LCPD a TPTS téměř totožné.

Nejvíce pravděpodobným a nejčastějším scénářem je, že v systémů bude mnohem více úkolů než dostupných vozidel. Tuto situaci simuluje poslední



Obrázek 5.2: Průměrný počet kroků na jeden úkol



Obrázek 5.3: Průměrná doba trvání úkolu

případ, kdy je v systému dvojnásobný počet úkolů než vozidel. Díky nejefektivnějšímu plánování je průměrná doba úkolu nejnižší opět u LCPD. Výsledek

TPTS se tentokrát spíše přibližuje nejhoršímu TP, protože efektivní rozdělování má zde již menší váhu než plánování. Podobně je tomu u průměrné krokové náročnost na jeden úkol. Všechny tři algoritmy využívala všechna vozidla stejnou dobu, ale algoritmus LCPD splnil nejvíce úkolů a to o 21 % více než TPTS a dokonce o 109 % více než TP.

5.3 Testování kolapsu systému

Každý dopravní systém má nějakou kapacitu úkolů, kterou dokáže zvládat. Při přesáhnutí této kapacity začne systém kolabovat. V tomto testování jsem zkoumal, jak určitý počet vozidel bude zvládat narůstající počet úkolů. V každé simulaci se proto postupem času zvyšuje počet nově příchozích úkolů. V každém pátém kroku přibude $\lceil \frac{t}{100} \rceil + 1$ úkolů, kde t je celkový počet kroků. Stav kolapsu systému jsem označil jako moment, kdy nějaký úkol čeká na vyřízení déle než 50 kroků. Pro zvyšující se počet dostupných vozidel jsem pro omezený prostor na mapě předpokládal, že výsledky budou podobné logaritmické křivce.

Výsledky simulací jsou na grafu na obrázku 5.4. U algoritmu TPTS můžeme pozorovat předpokládaný vývoj, kdy nejprve s přibývajícím počtem vozidel dochází k prodlužování doby do kolapsu systému a při počtu 65 dostupných vozidel dosáhne systém své maximální kapacity. Další výsledky se drží na podobné úrovni a algoritmus po většinu času nevyužíval další dostupná vozidla, protože pro ně nedokázal naplánovat validní trasu.

Výsledky algoritmu LCPD jsem naměřil pouze pro třetinu případů, protože pro větší počty vozidel jsem neměl dostatečný výpočetní výkon. Avšak předpokládám, že výsledky by byly podobné jako v případě TPTS, jak naznačují naměřené výsledky.

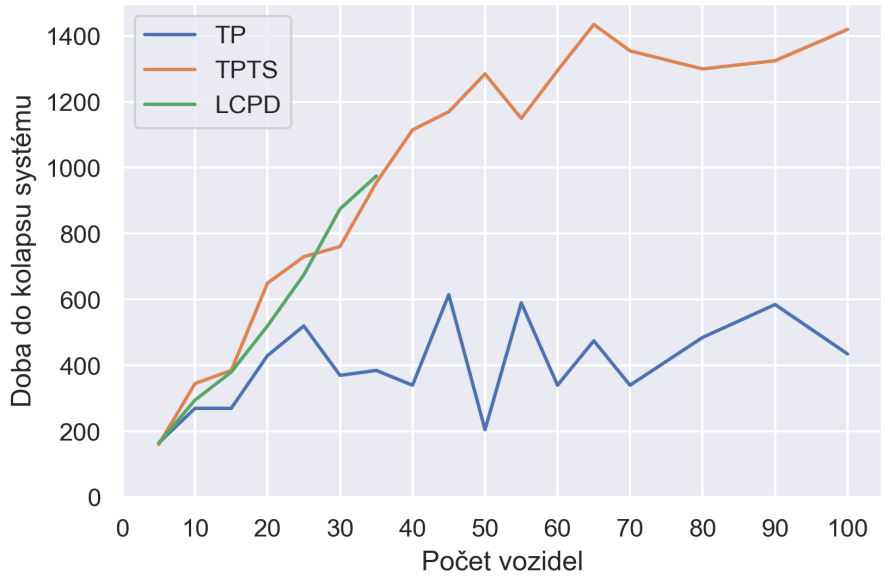
Algoritmus TP dosáhl kapacity systému mnohem dříve než ostatní algoritmy a jeho další skákající výsledky jsou způsobeny tím, jak dobře zrovna byla vozidla blízko urgentním úkolům.

Na obrázku 5.5 jsou zobrazeny výsledky simulací, kde na stejném scénáři byl sledován počet splněných úkolů do kolapsu systému. Při porovnání těchto dvou grafů je vidět korelace mezi dobou do kolapsu a počtem splněných úkolů.

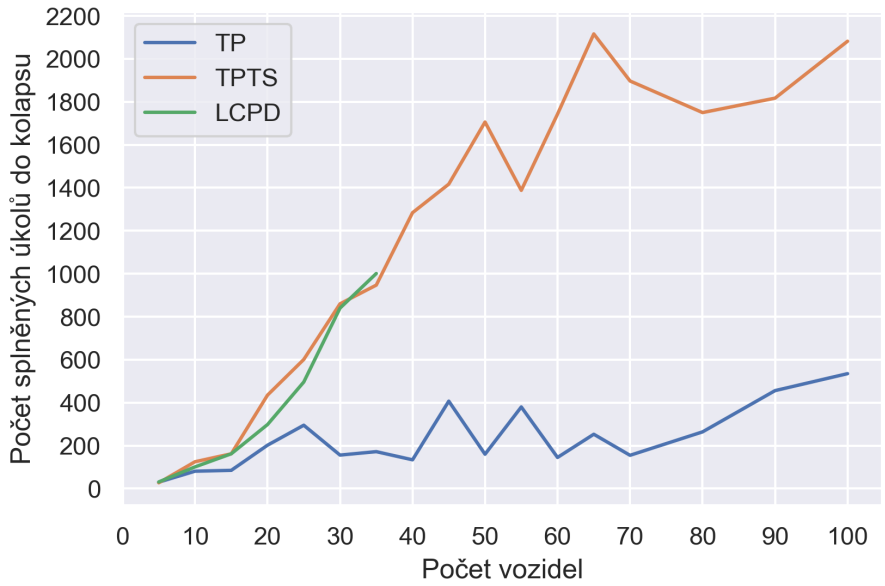
5.4 Efektivita plnění úkolů

V tomto scénáři jsem se zaměřil na zákazníkův pohled na systém. Aby byl zákazník spokojen, tak je zapotřebí včas reagovat na nově vzniklé požadavky. U úkolu lze porovnávat 2 doby, a to jak dlouho zákazník čeká než bude jeho úkol vyzvednut a také jak dlouhou dobu strávil celkově úkol v systému než byl dokončen.

Pro tuto simulaci jsem vytvořil scénář, kdy jsou v určité časy do systému přidávány nové úkoly. V celkovém průměru vzniká v každou časovou jednotku



Obrázek 5.4: Doba do kolapsu systému dle počtu vozidel

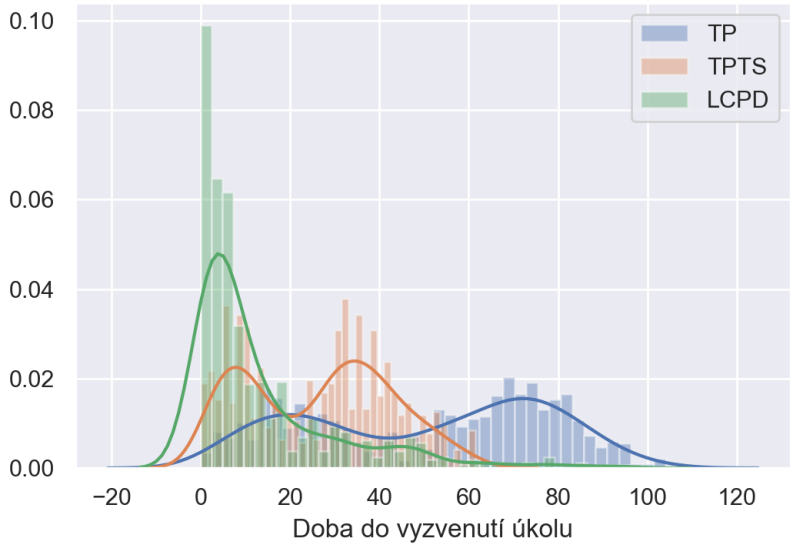


Obrázek 5.5: Počet splněných úkolů do kolapsu

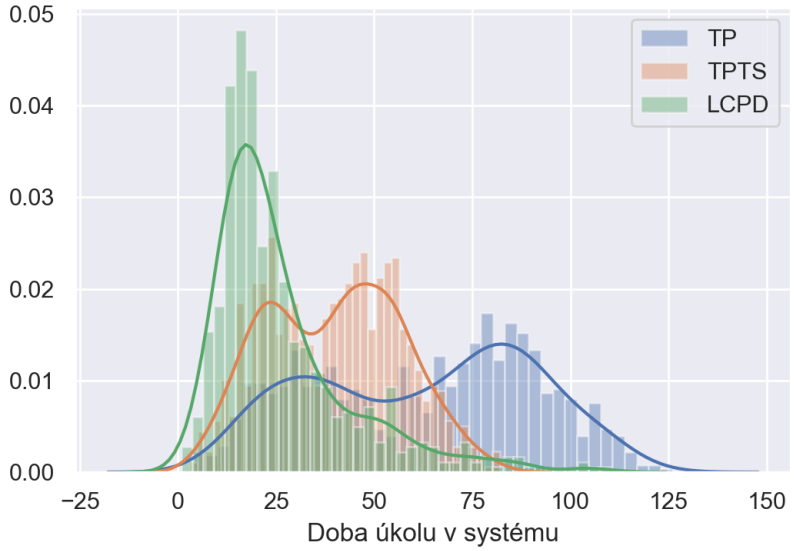
jeden úkol, avšak v prostřední třetině simulace je vytvářeno dvojnásobně více úkolů než ve zbývajících dvou třetinách. Tento scénář je beze změn testován

pro všechny tři algoritmy.

Za ideální výsledek lze považovat ten, kde naměřená doba trvání úkolu odpovídá normálnímu rozdělení s co nejmenší střední hodnotou a variancí.



Obrázek 5.6: Histogram doby do vyzvednutí úkolu



Obrázek 5.7: Histogram doby úkolu v systému

Výsledné histogramy se čtyřiceti přihrádkami jsou vidět na obrázku 5.6 pro dobu od vzniku úkolu do jeho vyzvednutí úkolu a obrázku 5.7 pro celkovou dobu úkolu v systému.

U doby do vyzvednutí úkolu má nejnižší medián algoritmus LCPD, a to 7. Nejvíce zastoupený interval je $< 0, 2.4)$ a je zde necelých 10 % všech hodnot. Pro další intervaly tato hodnota klesá. Z toho lze usuzovat, že vyřizovaní úkolů probíhá postupně podle toho, jak vstoupily do systému, a nedochází ke dlouhým čekáním.

Histogram doby do vyzvednuti pro TPTS vykazuje dva extrémy a jeho medián je 29. Doba čekání úkolu je v průměru delší než u LCPD. Hlavním problémem zde bude méně efektivní plánování tras, z čehož pramení delší čekací doby.

Algoritmus TP má čekací doby ještě více rozprostřené než předchozí dva algoritmy. Medián má zde hodnotu 56. Na jeho histogramu lze pozorovat rozdíl oproti TPTS. Toto zhoršení představuje pouze změna v rozdělování úkolů.

Histogramy dob, jak dlouho je úkol v systému, jsou podobné předchozímu případu. Algoritmy TP a TPTS mají opět větší rozptyl než LCPD a také více vrcholů, což může odpovídat změně frekvence přidávání úkolů do systému.

5.5 Vytíženost jednotlivých agentů

V tomto testování se zaměřuji na to, jak jednotlivé řídící algoritmy dokáží pracovat s určitým počtem dostupných vozidel. Nikde nelze předpokládat rovnoměrné využívání vozidel, ale výsledky simulace by se k tomu měly blížit a rozdíl mezi extrémy by neměl být příliš velký.

Pro toto testování jsem použil stejný scénář jako v kapitole 5.4, tedy totožný pro každý algoritmus. Při simulaci jsem pro každý dokončený úkol zaznamenal, jaké vozidlo ho doručilo. Následně jsem všechna vozidla spočítal jeho počet jejich úkolů. Výsledky jsou zaneseny do tabulky 5.1.

Tabulka 5.1: Výsledky využívání jednotlivých agentů dle počtu doručených úkolů

Algoritmus	Průměr	Směrodatná odchylka	Min	Max
TP	45,35	1,79	41	49
TPTS	45,35	2,20	41	49
LCPD	45,35	2,83	41	51

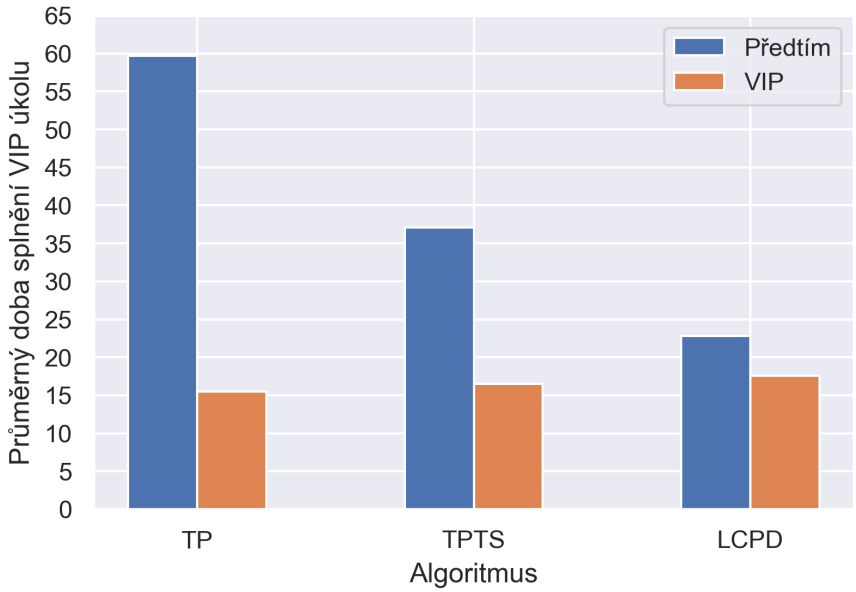
Průměrný počet dokončených úkolů je pro každý algoritmus stejný, protože byl pevně stanoven počet úkolů i vozidel. V čem se algoritmy liší, je směrodatná odchylka měření. Nejvíce spravedlivý je algoritmus TP, protože agenti dostávají nové úkoly až po dokončení předchozích a nedochází zde k výměně úkolů mezi agenty. Za cenu efektivnějšího řízení dopravy má vyšší směrodatnou odchylku

TPTS a nejvyšší LCPD. Rozdíl mezi vozidlem s nejmenším a největším počtem úkolů u algoritmu LCPD je 10 úkolů.

5.6 VIP požadavky

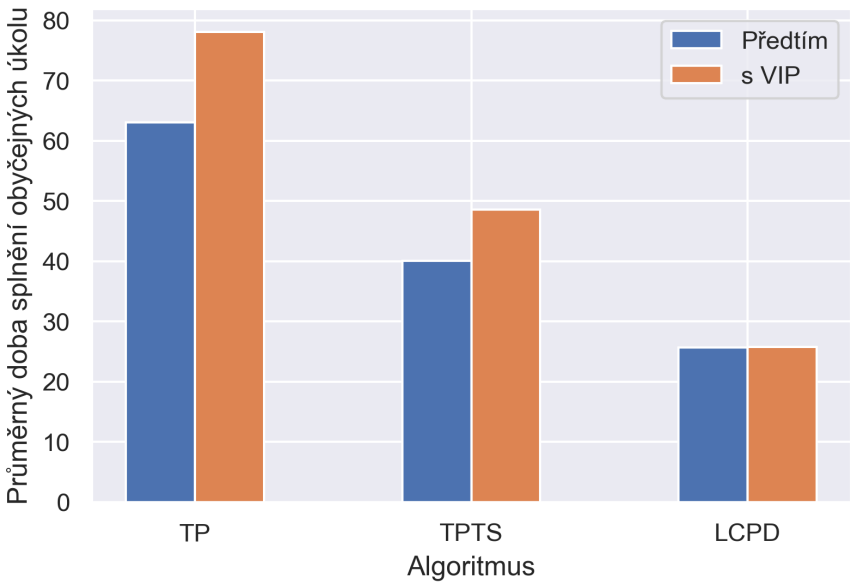
Jednou z faktorů, který velmi ovlivní celkovou efektivitu řízení dopravy je, jak se daný řídící algoritmus dokáže vypořádat s příchozími VIP požadavky. Snahou každého algoritmu by měla být minimalizace času potřebného pro dokončení VIP úkolu, ale také snaha příliš nenavýšit průměrnou dobu plnění obyčejných úkolů.

Pro toto testování jsem použil stejný scénář přidávání úkolů pro všechny tři algoritmy. Nejprve jsem provedl simulaci tak, že všechny úkoly měly stejnou prioritu a měřil jsem doby plnění úkolů a celkovou krokovou náročnost. Poté jsem spustil stejnou simulaci s tím, že každý desátý úkol měl VIP prioritu. Zaměřil jsem se na to, jak se změní výsledky pro odpovídající obyčejné úkoly v obou simulacích a jak pro VIP úkoly.

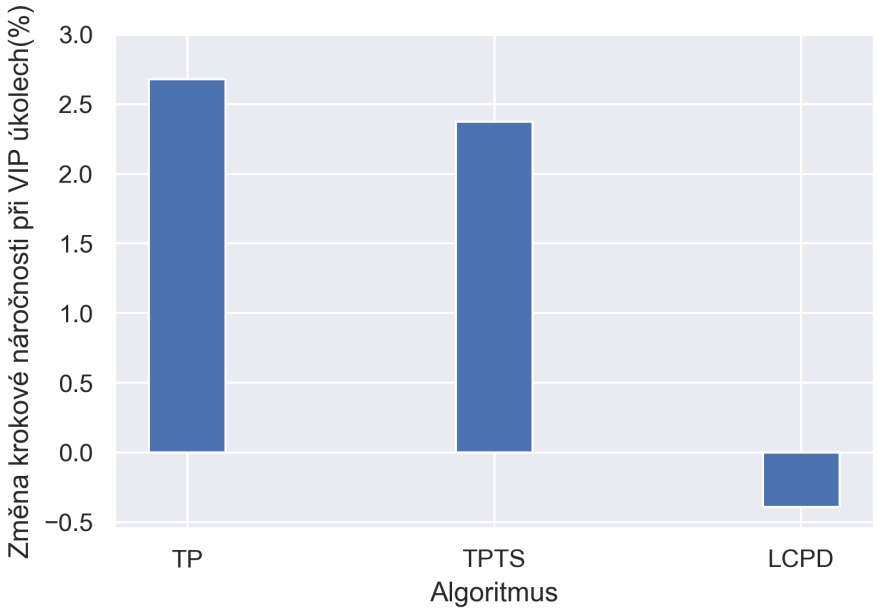


Obrázek 5.8: Průměrná doba plnění VIP úkolů

Na obrázku 5.8 je graf ukazující, jak se změnila průměrná doba plnění vybraných úkolů, které v první simulaci byly obyčejné a v druhé byly VIP. Velmi podobné hodnoty pro simulaci s VIP úkoly ukazují, že všechny tři algoritmy dokáží stejně zvládat VIP úkoly. Drobné rozdíly jsou způsobeny tím, jak daleko bylo nejbližší volné vozidlo pro obsloužení požadavku.



Obrázek 5.9: Průměrná doba plnění obyčejných úkolů



Obrázek 5.10: Procentuální změna krokové náročnosti při používání VIP úkolů

Pokud se ukázalo, že všechny tři algoritmy mají podobné průměrné časy pro VIP úkoly, je zapotřebí zjistit, jak se změnila průměrná doba pro ostatní

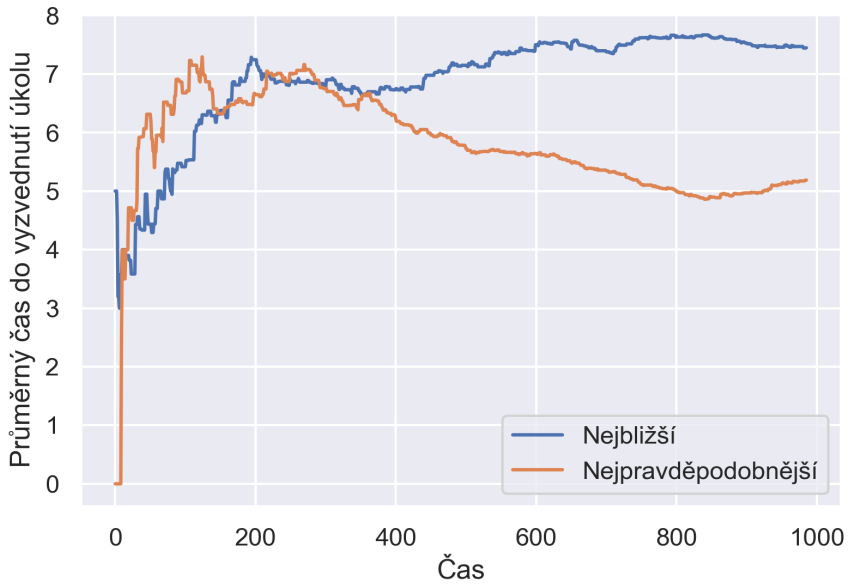
úkoly. Graf se změnami je na obrázku 5.9. U algoritmů TP a TPTS lze pozorovat zvýšení průměrné doby plnění obyčejných úkolů o zhruba 20 %. U algoritmu LCPD je to pouze o desetiny procenta.

Graf zobrazující změny v počtu kroků, které musí všechna vozidla vykonat, je na obrázku 5.10. Pokud tedy každý desátý úkol dostane vyšší prioritu než ostatní, tak se energetická náročnost systému nejvíce zvýší u algoritmu TP a TPTS. V tomto scénáři došlo u algoritmu LCPD dokonce ke snížení náročnosti. Toto spíše připisuji náhodě, kdy se vozidla po splnění VIP úkolu dostala na lepší pozici a byla blíže k vyzvedávacím pozicím nových úkolů.

5.7 Parkování dle výskytu úkolů

V tomto testování jsou porovnávány výsledky simulací, jak se změní průměrná vyzvedávací doba úkolu, pokud jsou vozidla parkována na nejbližší parkoviště, nebo na nejvhodnější, dle výskytu úkolů.

Ve scénáři tohoto testování nevznikají nové úkoly náhodně, nýbrž v určité oblasti je pravděpodobnost požadavku na vyzvednutí dvojnásobná oproti zbytku mapy. Pro testování byl použit algoritmus LCPD, 20 vozidel a 10 úkolů v každém čase.



Obrázek 5.11: Porovnání výsledků simulací při změně způsobu parkování

Na obrázku 5.11 je graf zobrazující výsledky simulací. Je zde vyobrazen průběh průměrné doby mezi příchodem úkolu do systému a jeho vyzvednutím vozidlem. V první třetině jsou výsledky pro oba přístupy velmi podobné, avšak

poté se začne projevovat úprava vah u parkovišť a průměrná doba začne pro tento nový přístup klesat. Po 1000 krocích takto klesla průměrná doba oproti standardnímu způsobu o 30 %. V důsledku ušetřeného času stihla vozidla také o 11 % více úkolů.

5.8 Výpočetní výkon

Klíčovým měřítkem, jak je daný algoritmus efektivně použitelný, jsou požadavky na výpočetní výkon. V kontextu dopravních systémů je zapotřebí uvažovat nad možnou škálovatelností podle počtu vozidel. Náročnost na výpočetní výkon by tedy ideálně měla růst lineárně s počtem dostupných vozidel.

Způsobů, jak výpočetní náročnost měřit, je několik. Rozhodl jsem se, že pro toto testování použiji jako ukazatel počet, kolikrát každý řídící algoritmus volá A*. Vzhledem k tomu, že každý algoritmus při volání A* prohledává stejný stavový prostor a implementace tohoto prohledávání je téměř stejná, jde o srovnatelné měřítko.

Scénář testování bude probíhat tak, že každý algoritmus bude postupně testován pro 10, 20 a 30 dostupných vozidel. V každý časový moment bude v systému stejný počet úkolů jako vozidel. Primárním cílem tohoto testování není poměřovat, jak si pro daný počet dostupných vozidel vedou algoritmy mezi sebou. Důležité je především jak se náročnost postupně mění u každého algoritmu zvlášť.

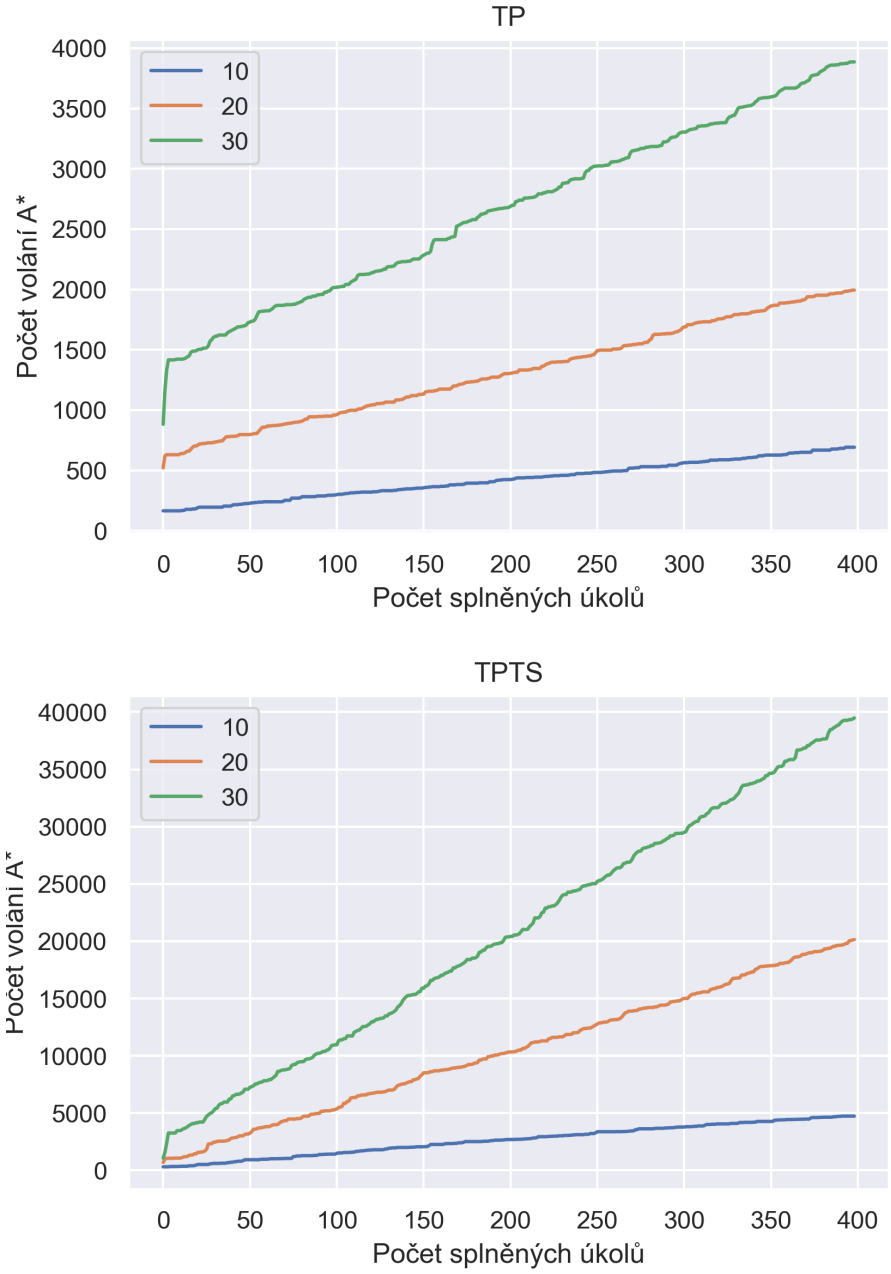
Výsledky simulace představují grafy na obrázku 5.12.

Počet volání A* u algoritmu TP se v průběhu času průběžně zvyšuje a nedochází zde k větším výkyvům. S rostoucím počtem vozidel dochází stále k větším potřebám na výpočetní výkon.

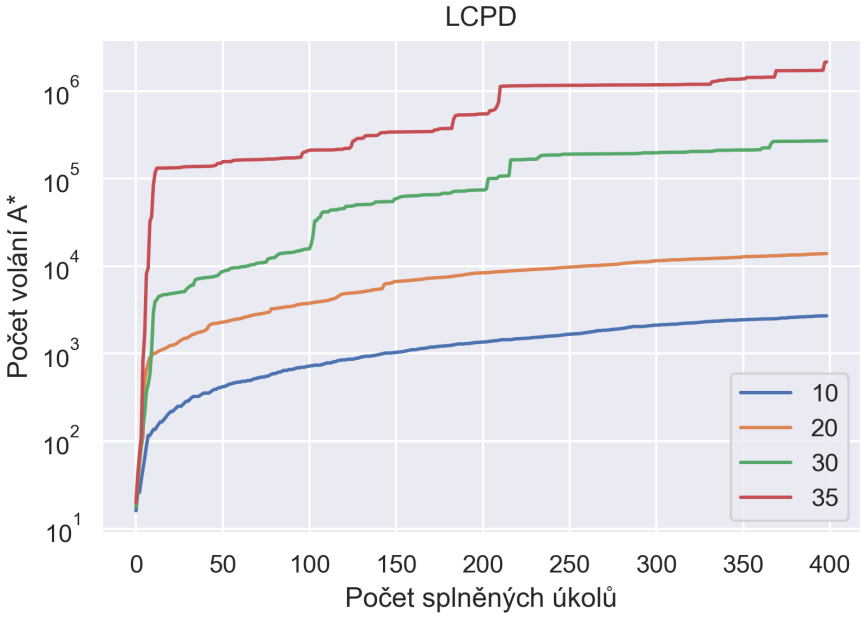
Výsledky algoritmu TPTS jsou velmi podobné jako TP. Je zajímavé, že pouze rozdílné přidělování úkolů zapříčinilo desetinásobný počet volání A* oproti TP.

Dle očekávání je s vyšším počtem vozidel nejvyšší potřeba volání A* právě pro algoritmus LCPD. Aby bylo možné jednotlivé simulace přehledně zobrazit, je v grafu na vertikální ose použito logaritmické měřítko. Pro 10 a 20 vozidel jsou stále výsledky lepší než u TPTS, avšak již pro 30 vozidel má LCPD více než šestkrát vyšší počet hledání tras. Pro srovnání jsem zde uvedl i výsledek pro 35 vozidel, kde je již velký skok oproti nižší variantě.

Hlavním problémem LCPD není celkový vysoký počet volání A*, ale že toto množství vzniklo převážně v několika určitých časech a ne rovnoměrně. Dle instance, kterou musí CBS řešit nelze předpovědět, jak dlouho nejspíše bude plánování trvat. Například u simulace se 35 vozidly dlouhé 400 časových kroků došlo v jednom kroku k tak velkému nárůstu počtu volání, že představoval 18 % výsledného počtu volání A*.



Obrázek 5.12: Vývoj počtu volání A* u řídících algoritmů dle počtu vozidel



Obrázek 5.12: Vývoj počtu volání A* u řídících algoritmů dle počtu vozidel

5.9 Shrnutí výsledků testování

Dle předpokladů měl ve většině testování nejlepší výsledky řídící algoritmus LCPD. Avšak potvrdilo se, že nároky na výpočetní výkon jsou u tohoto algoritmu příliš vysoké kvůli optimálnímu vyhledávání tras.

Zajímavé je, že výsledky algoritmu TPTS nepříliš zaostávají za LCPD. Pokud porovnáme horší výsledky ve většině testech s dobrými výsledky ve výpočetním výkonu, tak soudím, že výpočetní výkon má mnohem větší váhu.

Nejhorší výsledky měl algoritmus TP, který byl zařazen do testování pro svou jednoduchost a porovnání s ostatními algoritmy.

Závěr

Cílem této práce bylo navrhnout a otestovat možné algoritmy pro řízení autonomní dopravy ve městech. Po rešerši dopravních systémů a multi-agentních algoritmů jsem se rozhodl použít jako základ řídících algoritmů tyto tři algoritmy: Token Passing, Token Passing with Task Swaps a Lifelong Centralized Pickup and Delivery. Všechny tři algoritmy jsem upravil, aby je bylo možné použít pro řízení dopravy v teoretickém prostředí, které jsem definoval.

Jako vylepšení systému jsem navrhl a implementoval funkci pro přednostní odbavování VIP požadavků jako například průjezd vozidel integrovaného záchranného systému. Druhé vylepšení se týkalo vhodnějšího parkování vozidel v situacích, kdy požadavky nevznikají rovnoměrně po celém městě.

V testování jsem se zaměřil na schopnost algoritmů zvládat různé počty požadavků vůči počtu dostupných vozidel, na rozdíly v době plnění úkolů či na požadavky na výpočetní výkon a jejich růst.

Nejlepší výsledky při řízení dopravy vykazoval algoritmus Lifelong Centralized Pickup and Delivery, který využívá optimální plánovač tras. Avšak jeho požadavky na výpočetní výkon příliš rychle rostou s počtem vozidel a pro reálné použití by proto byl nejspíš nepoužitelný. Jeho přístup, kdy jsou trasy vozidel upravovány průběžně pro všechny agenty, shledávám správným a jeho největší nedostatek by možná mohl být vyřešen distribucí výpočtu, i za cenu mírně horších výsledků.

Přístup dalších dvou algoritmů neměl o tolik horší výsledky a mohl by být realizovatelný. Zde se ovšem dostáváme k problému, zda je možné v reálném prostředí přesně plánovat budoucnost a spoléhat se na ni.

Možným zlatým středem by mohl být algoritmus, který přeplánovává trasy všech vozidel najednou, avšak nesnaží se naplánovat celou trasu, ale jen několik příštích kroků ve směru optimální trasy a řeší tedy menší počet konfliktů.

Myslím, že až dojdeme k takovému technologickému pokroku, že bude možné využívat autonomní vozidla, tak centralizované řízení dopravy bude otázka, kterou se bude věda zabývat. Otázkou je, zda i s realizovatelným algoritmem by tento systém byl uskutečnitelný nebo je to pouhá utopie.

Bibliografie

1. BAQUELA, Enrique Gabriel; OLIVERA, Ana Carolina. Optimization of Traffic Network Design Using Nature-Inspired Algorithm: An Optimization via Simulation Approach. In: *Handbook of Research on Modern Optimization Algorithms and Applications in Engineering and Economics*. IGI Global, 2016.
2. BOLTZE, Manfred; ANH TUAN, Vu. Approaches to Achieve Sustainability in Traffic Management. *Procedia Engineering*. 2016, roč. 142, s. 204–211. Dostupné z DOI: 10.1016/j.proeng.2016.02.033.
3. CLARK, Larry. *Washington State University*. Dostupné také z: https://magazine.wsu.edu/web-extra/traffic-signals-a-brief-history/.
4. CLARK, Larry. *Smart signals*. Dostupné také z: https://magazine.wsu.edu/2019/08/02/smart-signals/.
5. TAEIHAGH, Araz; LIM, Hazel Si Min. Governing autonomous vehicles: emerging responses for safety, liability, privacy, cybersecurity, and industry risks. *Transport reviews*. 2019, roč. 39, č. 1.
6. CARPENTER, Shawn. *Autonomous Vehicle Radar: Improving Radar Performance with Simulation*. Dostupné také z: https://www.ansys.com/about-ansys/advantage-magazine/volume-xii-issue-1-2018/autonomous-vehicle-radar.
7. *International Releases Updated Visual Chart for Its "Levels of Driving Automation"Standard for Self-Driving Vehicles*. 2018. Dostupné také z: https://www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-%E2%80%9Clevels-of-driving-automation%E2%80%9D-standard-for-self-driving-vehicles.

8. *WP.29 Agreement 1968 Vienna Convention on Road Traffic*. Dostupné také z: https://globalautoregs.com/rules/157-1968-vienna-convention-on-road-traffic.
9. ANDERSON, James M; NIDHI, Kalra; STANLEY, Karlyn D; SORENSEN, Paul; SAMARAS, Constantine; OLUWATOLA, Oluwatobi A. *Autonomous vehicle technology: A guide for policymakers*. Rand Corporation, 2014.
10. *All countries compared for Transport ¿ Road ¿ Motor vehicles per 1000 people*. NationMaster. Dostupné také z: https://www.nationmaster.com/country-info/stats/Transport/Road/Motor-vehicles-per-1000-people.
11. HART, Peter E; NILSSON, Nils J; RAPHAEL, Bertram. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*. 1968, roč. 4, č. 2, s. 100–107.
12. STERN, Roni et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In: *Twelfth Annual Symposium on Combinatorial Search*. 2019.
13. SHARON, Guni; STERN, Roni; FELNER, Ariel; STURTEVANT, Nathan R. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*. 2015, roč. 219, s. 40–66.
14. WALKER, Thayne T.; STURTEVANT, Nathan R.; FELNER, Ariel. Extended Increasing Cost Tree Search for Non-Unit Cost Domains. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 2018, s. 534–540. Dostupné z DOI: 10.24963/ijcai.2018/74.
15. RATNER, Daniel; WARMUTH, Manfred K. Finding a Shortest Solution for the N× N Extension of the 15-PUZZLE Is Intractable. In: *AAAI*. 1986.
16. SILVER, David. Cooperative Pathfinding. *AIIDE*. 2005, roč. 1.
17. JANSEN, M. Renee; STURTEVANT, Nathan R. Direction Maps for Cooperative Pathfinding. In: *AIIDE*. 2008.
18. STANDLEY, Trevor Scott. Finding Optimal Solutions to Cooperative Pathfinding Problems. In: *AAAI*. 2010.
19. MA, Hang; LI, Jiaoyang; KUMAR, TK; KOENIG, Sven. Lifelong multi-agent path finding for online pickup and delivery tasks. *arXiv preprint arXiv:1705.10868*. 2017.
20. KUHN, Harold W. The Hungarian method for the assignment problem. *Naval research logistics quarterly*. 1955, roč. 2, č. 1-2, s. 83–97.

Příloha A

Implementace

Tato kapitola stručně popisuje, jakým způsobem došlo k implementaci uvedených algoritmů a simulačního prostředí. Pro implementaci byl zvolen programovací jazyk Python ve verzi 3.8.

Každý ze tří řídících algoritmů je představován svou vlastní třídou. Vstupem je vždy čtveřice:

množina vozidel – Vozidlo má podobu třídy se všemi atributy a metodami dle stavového systému.

mapa – Uvnitř této třídy je bitmapa, která definuje na jakých pozicích se nachází silnice, parkoviště či zástavba.

množina aktuálních úkolů – Do této množiny se postupně přidávací úkoly pomocí reference. Samotný úkol nese informace o své vyzvedávací a doručovací pozici, indikátorem, jestli se jedná o VIP, a časy, kdy nastalo k daným změnám.

množina parkovišť – Parkoviště slouží pro přidávání a odebírání aut ze systému. Je nutné definovat pozici parkoviště a počáteční počet vozidel.

Všechny řídící algoritmy mají metodu se stejným názvem – *do_step*, která slouží k naplánování pro aktuální časový krok. Je tedy možné použitý algoritmus libovolně měnit.

Úkoly mohou vznikat dvěma způsoby – náhodně či staticky. V případě náhodného přístupu jsou vždy uvažovány všechny body mapy se silnicí kromě křižovatek a náhodným způsobem jsou dva odlišné body vybrány jako vyzvedávací a doručovací pozice. Statickým přístupem jsou úkoly na začátku simulace načteny z CSV souboru o čtyřech sloupcích – id úkolu, vyzvedávací pozice, doručovací pozice a čas, kdy přidat úkol do systému. Pozice jsou ve formátu "(y, x)".

Před spuštěním simulace je tedy potřeba definovat tyto věci:

- mapa,
- parkoviště,
- řídící algoritmus a
- způsob vzniku úkolů.

Grafické rozhraní vzniklo pomocí frameworku Pygame. Mapa je definována počtem plánovacích úseků (ve tvaru čtverce) v řádku a sloupci. Každá část mapy má vzhled, který se definuje dle okolních bodů. Při pohybu mezi plánovacími úseky se vozidlo posouvá po malých vzdálenostech a díky častému vykreslování je toto vnímáno jako plynulý pohyb.

Příloha B

Seznam použitých zkratek

SAT Boolean Satisfiability Problem

CSP Constraint Satisfaction Problem

ASP Answer Set Programming

CBS Conflict-Based Search

CT Constraint Tree

MAPD Multi-Agent Pickup and Delivery

TP Token Passing

TPTS Token Passing with Task Swaps

LCPD Lifelong Centralized Pickup and Delivery

MAPF Multi-Agent Pathfinding

Příloha C

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis.tex` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF