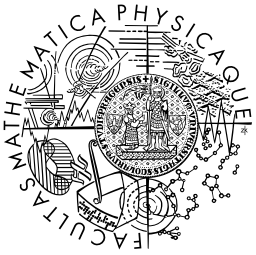


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Klára Tauchmanová

Plánování cest pro multi-robotické sklady

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: prof. RNDr. Barták Roman, Ph.D.

Studijní program: Informatika

Studijní obor: Obecná informatika

Praha 2021

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Název práce: Plánování cest pro multi-robotické sklady

Autor: Klára Tauchmanová

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: prof. RNDr. Barták Roman, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: Práce se zabývá problémem hledání bezkolizních cest v automatizovaném skladu pro skupinu agentů, tzv. Multi-agent pickup and delivery (MAPD) problémem. Agentům ve skladu jsou kontinuálně zadávány nové úkoly, které vyžadují vyzvednout zboží na určeném místě a převézt ho na jiné místo v prostředí skladu. V práci je popsán hierarchický algoritmus na řešení MAPD problémů, který provádí plánování na dvou úrovních - globální a lokální. Algoritmus pracuje s prostředím skladu, které je ručně rozděleno na oblasti. Na globální úrovni dochází k zadávání úkolů volným agentům a následně k plánování posloupnosti oblastí, které agent při vykonávání úkolu navštíví. Úkol je zadán vždy nejbližšímu volnému agentovi. Na lokální úrovni dochází k samotnému hledání bezkolizních cest v rámci jednotlivých oblastí, ke kterému se používají upravené Multi-agent path finding (MAPF) algoritmy. Navržený algoritmus byl otestován v simulovaném prostředí. Experimentálně jsme ukázali, že hierarchický algoritmus používající Conflict-based search (CBS) na lokální úrovni má menší runtime a nachází řešení s menším makespanem než algoritmus využívající redukci na SAT. Dále ukazujeme, že hierarchický přístup k řešení MAPD výrazně snižuje runtime algoritmu, zatímco nalezená řešení nejsou o mnoho horší než u nehierarchického přístupu.

Klíčová slova: Multi-agent pickup and delivery, Dvojúrovňové plánování, Multi-agentní hledání cest, Automatizované sklady

Title: Path planning for multi-robot warehouses

Author: Klára Tauchmanová

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: prof. RNDr. Barták Roman, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Thesis covers the problem of finding non-colliding paths in automated warehouses for a group of agents - so called Multi-agent pickup and delivery (MAPD) problem. Agents in the warehouses are constantly engaged with new tasks which require picking up the object on certain coordinates and transporting it to another place in the warehouse. Thesis describes a hierarchical algorithm for MAPD problems that plans paths in two levels - global and local. The algorithm works with a warehouse environment manually divided into sectors. Planning on the global level consists of assigning tasks to free agents and selecting a sequence of sectors to go through. The task is always assigned to the closest free agent. On the local level, the actual path through the sector is planned using modified Multi-agent path finding (MAPF) algorithms. The proposed algorithm was tested in a simulated environment. We showed that the hierarchical algorithm with Conflict-based search (CBS) on the local level has a smaller runtime and finds a solution with a lower makespan than the algorithm using reduction to SAT. Further, we showed that the hierarchical approach for MAPD significantly lowers the runtime of the algorithm, meanwhile, the outputted solutions are not much worse than for non-hierarchical approaches.

Keywords: Multi-agent pickup and delivery, Two-level planning, Multi-agent path finding, Automated warehouses

Obsah

Úvod

S postupem globalizace začala hrát logistika klíčovou roli prakticky ve všech oblastech podnikového řízení. Pojem logistika v sobě zahrnuje jak organizaci, tak i plánování a řízení toků zboží, informací, služeb či dalších zdrojů. Cílem je zajistit, že bude požadovaný zdroj ve správném množství a kvalitě, ve správný čas na správném místě, a to pokud možno s co nejmenšími náklady. Jednou z podstatných součástí logistického systému je skladování a tzv. skladová logistika, na jejíž zefektivňování je v dnešní době kladen veliký důraz. Zefektivňování skladových (a dalších) procesů bývá úzce svázané s automatizací, která představuje moderní a úsporné řešení problému. Automatizace skladových procesů zrychluje pohyb zboží, a tím zvyšuje celkovou efektivitu skladu, snižuje náklady a zároveň přispívá ke zvýšení bezpečnosti skladu. V automatizovaných skladech se pohybují autonomně řízení roboti, kteří převážejí zboží z jednoho místa na druhé. Roboti mají typicky za úkol vyzvednout zboží z regálu a dopravit ho na výdejní místo, kde je zboží následně manuálně připraveno k dalšímu zpracování. Zapojení robotů ve skladech s sebou ale přináší problém s jejich vzájemnou koordinací – není žádoucí, aby docházelo ke kolizím mezi roboty. Problém koordinace robotů se často řeší v abstraktnější podobě. Prostředí skladu bývá modelované rovinným grafem a místo robotů se pracuje s nehmotnými agenty, kteří se pohybují po vrcholech grafu.

Abstraktní formulaci problému koordinace robotů můžeme modelovat jako tzv. Multi-agent path finding (MAPF) problém, což je problém hledání bezkolizních cest v multi-agentním prostředí. V MAPF máme pevně danou skupinu agentů, která se společně nachází ve známém prostředí. Každému agentovi je přiřazena jedna cílová lokalita. Úkolem je dopravit všechny agenty do jejich cíle tak, aby nedošlo ke kolizi mezi žádnými dvěma agenty. Přesná definice kolize záleží na konkrétní aplikaci problému, ale většinou se snažíme zabránit tomu, aby se dva agenti nacházeli ve stejném vrcholu (a tedy na stejném místě ve skladu).

Pokud se na automatizovaný sklad díváme v jeden konkrétní okamžik, pak lze situaci ve skladu docela dobře popsat jako nějaký MAPF problém. Každý agent reprezentující jednoho robota má přiřazenou lokaci, kam musí dojet – buď robot jede vyzvednout zboží z regálu (a jeho cíl je určen pozicí regálu), nebo naopak dopravuje již vyzvednuté zboží na výdejní místo (a jeho cíl pak představuje výdejní místo). Obecně ale problém skladu jako MAPF popsat nedokážeme. V automatizovaném skladu jsou totiž robotům kontinuálně zadávány nové úkoly, které mají vykonat. Průběžně tedy musí docházet jak k přiřazování nesplněných úkolů jednotlivým robotům, tak i k hledaní bezkolizních cest pro všechny roboty ve skladu. Navíc se předpokládá, že každý robot může v jeden čas vykonávat pouze jeden úkol, i když jeden úkol může představovat vyzvednutí zboží z více různých regálů a následné dopravení na výdejní místo. Problém automatizovaného skladu se často modeluje jako tzv. Multi-agent pickup and delivery (MAPD) problém. Problém MAPD představuje zobecnění problému MAPF.

Cílem této práce je naimplementovat algoritmus na řešení MAPD problémů v prostředí automatizovaného skladu.

Struktura práce

Nejprve si problém Multi-agent pickup and delivery (MAPD) formálně zadefinujeme a představíme si jeho varianty. V další kapitole MAPD problém zasadíme do kontextu MAPF problémů a podíváme se na jednotlivé přístupy, se kterými se můžeme při řešení MAPD v literatuře setkat. Následně se blíže seznámíme se dvěma známými MAPF algoritmy, jejichž modifikace budeme při řešení MAPD používat. V kapitole 4 podrobně popíšeme hierarchický algoritmus na řešení MAPD problémů. Algoritmus bude vycházet z principu hierarchického plánování z článku Liu a kol. (2019b). V poslední kapitole vyzkoušíme hierarchický algoritmus v simulovaném prostředí a empiricky prozkoumáme jeho vlastnosti.

1. Definice problému

Začneme uvedením formální definice Multi-agent pickup and delivery (MAPD) problému a několika dalších souvisejících pojmů.

1.1 MAPD problém

V MAPD se množina agentů pohybuje v předem známém prostředí. Prostředí můžeme reprezentovat grafem, jehož vrcholy představují potenciální pozice agentů a hrany znázorňují možné přechody mezi pozicemi. Agentům jsou v průběhu času zadávány úkoly, která mají vykonat. Úkol spočívá v převezení zboží z jednoho místa na jiné.

Definice 1 (Liu a kol. 2019a)**.** ***Instance MAPD problému*** *je trojice* (G, A, T)*, kde* $G = (V,E)$ *je neorientovaný graf reprezentující společné prostředí,* $A = \{a_1,a_2,\dots,a_m\}$ *je množina agentů a* $T = \{t_1,t_2,\dots\}$ *je množina úkolů. Agent* $a_i \in A$ *je reprezentovaný unikátní počáteční pozicí* $s_i \in V$*. Úkol* $t_j \in T$ *je charakterizován trojicí* (r_j,s_j,g_j)*, kde* $r_j \in \mathbb{N}$ *je čas zveřejnění,* $s_j \in V$ *je místo vydání a* $g_j \in V$ *je místo doručení.*

Agenti se pohybují po grafu G v diskrétních časových krocích. V každém kroku se agent buď přesune na sousední políčko v G, nebo zůstane na své aktuální pozici. Obě akce mají jednotkovou délku trvání. Agenti se musí při svých přesunech vyhýbat kolizím s ostatními agenty, přičemž za kolizi považujeme kolizi ve vrcholu a kolizi na hraně.

Definice 2 (Ma a kol. 2017)**.** *Nechť* $l_i(t) \in V$ *označuje pozici agenta* $a_i \in A$ *v čase* t*.* ***Kolize ve vrcholu*** *nastane ve chvíli, kdy se dva agenti nacházejí ve stejný čas ve stejném vrcholu. Tedy* $\exists a_i,a_j \in A, a_i \neq a_j : l_i(t) = l_j(t)$ *pro nějaké* t*. Ke* ***kolizi na hraně*** *dojde v momentě, kdy se dva agenti pohybují po stejné hraně v opačném směru, neboli* $\exists a_i,a_j \in A, a_i \neq a_j : l_i(t) = l_j(t+1) \wedge l_j(t) = l_i(t+1)$ *pro nějaké* t*.*

Agent je volný, pokud nemá přiřazený žádný úkol. Je-li agentovi a_i přiřazen úkol t_j, musí se agent přesunout ze své aktuální pozice přes místo vydání s_j na místo doručení úkolu g_j. Ve chvíli, kdy agent dorazí na místo doručení úkolu, je úkol dokončen a agent je opět volný. Formálně můžeme čas dokončení úkolu zadefinovat následovně:

Definice 3. ***Úkol*** $t_j \in T$ ***je dokončen*** *v čase* $p_j \in \mathbb{N}$*, pokud existuje agent* $a_i \in A$*, který se po čase zveřejnění úkolu* r_j *nacházel v místě vydání úkolu* s_j *a který se v čase* p_j *nacházel v místě doručení úkolu* g_j*. Tedy* $\exists a_i \in A : l_i(k) = s_j \wedge l_i(p_j) = g_j$ *pro nějaké* $k \geq r_j$*. Čas dokončení úkolu* j *budeme označovat* p_j*.*

Definice 4. *Označme* t_{max} *čas, ve kterém dojde k dokončení posledního nedokončeného úkolu.* ***Plánem agenta*** a_i *rozumíme posloupnost pozic, na kterých se agent* a_i *nacházel v časech* t *pro* $t = 0,\dots,t_{max}$*.* ***Plán*** *agenta* a_i *je* ***validní****, pokud* $l_i(0) = s_i$ *a pro každé* $j \in \{0,\dots,t_{max}-1\}$ *platí, že* $(l_i(j),l_i(j+1)) \in E$*.*

Cílem MAPD je dokončit všechny úkoly. Řešením problému je množina plánů pro jednotlivé agenty. Aby bylo řešení korektní, musí být všechny dílčí plány validní, stejně dlouhé a v žádném čase nemůže dojít ke kolizi žádných dvou či více agentů. Problém je vyřešený, pokud jsou všechny úkoly dokončeny po konečném množství kroků.

1.1.1 Online vs. offline

V literatuře se rozlišují dvě verze MAPD problému - online a offline. V offline verzi MAPD je množina úkolů T známa předem a tato informace se využívá v průběhu výpočtu. Naopak v online verzi se úkoly zveřejňují průběžně. Místo množiny T se v online verzi pracuje s množinou τ, která obsahuje všechny zveřejněné úkoly. V čase t bude množina τ obsahovat všechny úkoly j, jejichž čas zveřejnění $r_j \leq t$. Online systém tedy pracuje s neúplnou informací.

V této práci se budeme zabývat online verzí problému.

1.1.2 Metriky

K porovnávání algoritmů potřebujeme umět zhodnotit kvalitu řešení. Mezi nejčastější kritéria pro porovnávání patří minimalizace *makespanu* a minimalizace *service timu*.

Definice 5 (Service time)**.** *Nechť $f(t_i)$ značí čas dokončení i-tého úkolu. Potom service time řešení definujeme jako*

$$ScT = \frac{1}{|T|} \sum_{t_i \in T} (f(t_i) - r_i).$$

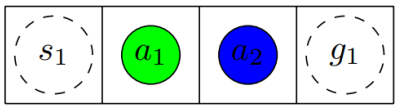
Definice 6 (Makespan)**.** *Nechť $f(t_i)$ značí čas dokončení i-tého úkolu. Potom makespan řešení definujeme jako*

$$Mks = \max_{t_i \in T} f(t_i) - \min_{t_i \in T} r_i.$$

Na *makespan* můžeme nahlížet jako na celkovou dobu v krocích, kterou potřebujeme k dokončení všech úkolů. *Service time* pak vyjadřuje propustnost systému – jak rychle jsou v průměru jednotlivé úkoly vykonány po jejich příchodu do systému.

1.2 Dobře formovaná instance

Ne všechny instance MAPD jsou řešitelné. Obrázek 1.1 zobrazuje MAPD instanci se dvěma agenty a_1 a a_2 a jedním úkolem t_1, který se má doručit z pozice s_1 na pozici s_2. Tato instance není řešitelná, neboť žádný z agentů nemůže dokončit úkol t_1. V literatuře se často zavádí pojem *dobře formované instance* („well-formed instances“ v originále, Ma a kol. (2017)), který pomocí dodatečných podmínek zaručí řešitelnost MAPD instancí. Dobře formované instance MAPD jsou podmnožinou všech MAPD instancí.



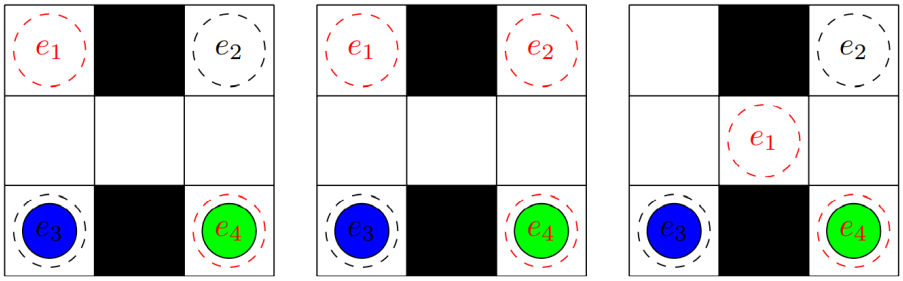
Obrázek 1.1: Příklad neřešitelné instance MAPD (Ma a kol., 2017)

Definice 7 (Dobře formovaná instance)**.** *Nechť množina* ***koncových bodů úloh*** *označuje množinu všech míst vydání a míst doručení úkolů a nechť množina* ***koncových bodů agentů*** *označuje množinu všech počátečních pozic agentů a případnou množinu speciálně vyhrazených parkovacích lokací. Pojmem* ***koncové body*** *označme sjednocení množiny koncových bodů úloh a množiny koncových bodů agentů. Instance MAPD je dobře formovaná, pokud*

1. *počet úkolů je konečný,*
2. *mezi každými dvěma koncovými body existuje cesta, která neprochází žádným dalším koncovým bodem,*
3. *počet koncových bodů agentů musí být alespoň tak velký, jako je počet agentů.*

Na obrázku 1.2 jsou znázorněny tři MAPD instance. Černá políčka představují překážky, zelený a modrý kruh reprezentují počáteční pozice dvou agentů. Černé přerušované kruhy znázorňují koncové body agentů, červené přerušované kruhy značí koncové body úloh. První instance je dobře formovaná, neboť splňuje všechny tři podmínky. Druhá instance porušuje (3), protože obsahuje pouze jeden černý přerušovaný kruh. Třetí instance zase porušuje (2).

Ma a kol. (2017) ve svém článku dokázali, že každá dobře formovaná instance je řešitelná. V této práci tedy budeme pracovat pouze s dobře formovanými instancemi.



Obrázek 1.2: Tři MAPD instance, první je dobře formovaná, druhá a třetí není (Ma a kol., 2017)

2. Související práce

V této kapitole se podíváme na to, jak MAPD problém souvisí s tzv. Multi-agent path finding problémem. Dále si pak ukážeme, s jakými přístupy se při řešení MAPD můžeme v literatuře setkat.

Multi-agent path finding (MAPF) je problém plánování takových cest pro množinu agentů, že pokud agenti pojedou po naplánované cestě, tak do sebe žádní dva agenti nenarazí. Existují různé varianty MAPF problému lišící se v tom, jaké mají předpoklady o agentech a o prostředí, ve kterém se agenti pohybují. Stern a kol. (2019) zavedli jednotnou terminologii vztahující se k MAPF a popsali varianty MAPF problému vyskytujícího se v odborné literatuře.

Jednou z variant MAPF, která je zároveň přirozeným rozšířením klasického MAPF problému, je tzv. **online MAPF problém**. Obecně se jedná o sekvenci MAPF problémů s proměnlivým počtem úkolů nebo agentů. Online MAPF můžeme dále rozdělit na problém MAPD a problém křižovatky.

Problém křižovatky je varianta MAPF problému pracující s proměnlivým počtem agentů. Každý z agentů má pouze jediný úkol - dostat se na svou cílovou pozici. Oproti klasickému MAPF se ovšem nepracuje s fixní množinou agentů, ale množina agentů se v průběhu času mění - nový agent může do systému vstoupit kdykoliv. Problémem křižovatky se ve svém článku zabývali Švancara a kol. (2019).

Problém MAPD je inspirován autonomními sklady, ve kterých roboti převážejí zboží z jednoho místa na jiné. Pracuje se s pevně danou množinou agentů, která se pohybuje v prostředí skladu a vykonává sekvenci doručovacích úkolů. Každý úkol má definovaný čas zveřejnění, což je čas, od kterého je možné daný úkol začít vykonávat. Jak již bylo zmíněno v předchozí kapitole, rozlišujeme dvě verze MAPD problému - online a offline. V offline verzi jsou všechny úkoly a jejich časy zveřejnění známé předem. Naopak v online verzi se úkoly zveřejňují průběžně, což znamená, že se systém o existenci novém úkolu dozví až v čase jeho zveřejnění.

V následujících sekcích se blíže podíváme na jednotlivé přístupy k řešení MAPD problémů.

2.1 Offline přístup

První přístup vychází z pozorování, že v mnoha reálných situacích se o existenci úkolu dozvíme dříve, než bude možné úkol začít vykonávat. Příkladem mohou být automatizované sklady, ve kterých musí být balíky nejprve připraveny a až následně mohou být agenty přepraveny.

Nguyen a kol. (2017) přeformulovali offline MAPD problém jako problém answer set programming. Liu a kol. (2019a) představili dva offline algoritmy, TA-Prioritized a TA-Hybrid, které využívají znalost úkolů k naplánování co nejvýhodnějšího rozdělení úkolů mezi agenty. Oba algoritmy na počátku rozdělí všechny úkoly mezi agenty pomocí vyřešení modifikovaného problému obchodního cestujícího a následně hledají bezkolizní cesty pro agenty se zadanou sekvencí úkolů. Algoritmy se pak od sebe vzájemně odlišují ve způsobu plánování cest.

2.2 Online přístup

Online algoritmy se na MAPD problém dívají jako na sekvenci MAPF problémů, přičemž při řešení MAPF se za počáteční pozici agenta považuje jeho aktuální poloha. K řešení jednotlivých MAPF instancí se využívají modifikované MAPF algoritmy. Jednotlivé online algoritmy se liší v tom, jak často a pro které agenty cesty přeplánovávají.

Příkladem algoritmu, který každou jednotku času přeplánovává cestu pro všechny agenty, je inkrementální Conflict-based search (CBS) (Wan a kol., 2018). Tento algoritmus se snaží o znovupoužití stromu omezení z běhu algoritmu v předchozím kroku. Strom omezení je popsán v sekci 3.1.

Další možností je přeplánovávat cesty každou časovou jednotku, ale pouze pro agenty, kteří dorazili na konec naplánované cesty. Tento přístup má ale dvě hlavní nevýhody - může být neúplný a často generuje drahá řešení. Na MAPD prostředí jsou z toho důvodu často kladeny dodatečné podmínky, které zaručí úplnost. Příkladem tohoto přístupu jsou algoritmy Token Passing (TP), Token Passing with Task Swaps (TPTS) (Ma a kol., 2017) a jejich vylepšená verze SIPPwRT (Ma a kol., 2018). Všechny tři algoritmy jsou založeny na principu předávání tokenu. Token je sdílený blok paměti, který obsahuje aktuálně naplánované cesty pro všechny agenty, množinu nezadaných úkolů a informace o přiřazení agentů k úkolům. V každém časovém kroku si všichni volní agenti zažádají systém o token. Systém pak postupně přepošle token všem agentům, kteří si o něj požádali. Agent držící token se za určitých podmínek může sám přiřadit k jednomu z dosud nezadaných úkolů. Pokud se tak stane, agent si naplánuje cestu tak, aby nekolidovala s žádnou jinou cestou uloženou v tokenu, a naplánovanou cestu uloží do tokenu. Nakonec vrátí token systému. Algoritmus TPTS se od TP liší tím, že umožňuje jednomu agentovi převzít úkol, který byl přiřazen jinému agentovi, pokud daný úkol ještě nezačal být vykonáván. Úkol začal být vykonáván, pokud agent, který je k danému úkolu přiřazený, dorazil na místo vydání úkolu.

Li a kol. (2020) přišli s algoritmem, který přeplánovává cesty všem agentům jednou za h časových jednotek (h definuje uživatel). Tento algoritmus pracuje s modifikovanou instancí MAPF (tzv. windowed MAPF), která řeší konflikty mezi agenty pouze v prvních w časových krocích algoritmu.

Online MAPD algoritmy mohou být použity na řešení offline varianty MAPD, ale vzhledem k tomu, že online algoritmy většinou nevyužívají všechny dostupné informace, bývají méně efektivní.

2.3 Hierarchický přístup

Oba dva výše zmíněné přístupy mají jeden společný problém - špatně se škálují na větší MAPD instance. Online algoritmy sice zvládnou vyřešit o něco větší instance než offline algoritmy, ale přeplánovávání cest každou časovou jednotku je pořád časově náročné. Problém se škálovatelností ale není problémem specifickým pouze pro MAPD. Tímto problémem trpí jak MAPF problémy, tak i samotný problém hledání cest. A právě při snaze o řešení problému škálovatelnosti vznikla myšlenka hierarchického plánování, která vychází z principu rozdělení většího problému na menší podproblémy.

Hierarchické plánování bylo navrženo za účelem snížení časové náročnosti problému hledání cest v komerčních počítačových hrách. Jedním z algoritmů pro hierarchické plánování cest je algoritmus HPA*, který představili Botea a kol. (2004). Tento algoritmus nejprve rozdělí mapu na oblasti a následně praktikuje hledání cest na dvou úrovních - globální a lokální. Výsledkem hledání cesty na globální úrovni je posloupnost oblastí, přes které cesta prochází. Na lokální úrovni se pak hledá optimální cesta v rámci jednotlivých oblastí. Pro hledání cest se na obou úrovních používá algoritmus A* (Hart a kol., 1968).

Koncept hierarchického plánování aplikovaný na řešení MAPF problému představili ve svém článku Pianpak a kol. (2019). Algoritmus pracuje s mapou rozdělenou na oblasti. Každá oblast má přidělený jeden řešič, který zodpovídá za plánování cest pro agenty v dané oblasti. Plánování cest pak probíhá ve dvou fázích. V první fázi dochází k plánování abstraktních cest a domlouvání přesunu agentů mezi jednotlivými oblastmi. Abstraktní cestou agenta se rozumí posloupnost řešičů, jejichž oblasti agent během své cesty do cíle navštíví. Tato cesta se plánuje pomocí Breadth-first search (Prohledávání do šířky) (BFS). Během druhé fáze se řeší samotný problém plánování bezkolizních cest v jednotlivých oblastech. K plánování cest v rámci oblasti se využívá převod MAPF problému na Answer set programming (ASP) (Gelfond a Lifschitz, 1988).

Liu a kol. (2019b) ve své práci nastínili způsob použití hierarchického plánování pro řešení online verze MAPD problémů. Mapa, se kterou algoritmus pracuje, je rozdělená na několik oblastí. Každá oblast musí splňoval několik omezení, takových, aby bylo možné později zaručit řešitelnost problému na lokální úrovni. Protože algoritmus řeší online verzi MAPD problému, jsou jednotlivé doručovací úkoly zveřejňovány průběžně. Na globální úrovni dochází k zadávání nově zveřejněných úkolů volným agentům a následně jsou pro tyto agenty naplánovány cesty v grafu skladu. Graf skladu je ohodnocený graf, jehož vrcholy reprezentují jednotlivé sektory a jehož hrany představují přejezdy mezi oblastmi. Cena hrany je dána vzdáleností středů oblastí, počtem abnormálních agentů (agenti zpoždění oproti svému plánu či agenti, kterým nějak selhala komunikace) a aktuální hustotou dopravy v druhé oblasti. Pro plánování cesty na globální úrovni se používá algoritmus A*. Přiřazování úkolů se dělá hladově – úkol se vždy zadá nejbližšímu volnému agentovi. Algoritmus se snaží o vybalancování dopravy v jednotlivých oblastech, čímž snižuje pravděpodobnost kolizí a zvyšuje celkovou efektivitu autonomního skladu. Na lokální úrovni probíhá samotné plánování bezkolizních cest pro jednotlivé oblasti. Plánování na této úrovni je decentralizované – každá oblast má svého plánovače, takže plánování může probíhat paralelně. Plánovač pracuje s orientovaným grafem oblasti, který reprezentuje jednosměrné silnice mezi regály ve skladu. Jednosměrnost silnic umožňuje zvýšit rychlost provozu a snížit množství dopravních zácp. Bezkolizní cesty jsou hledány pomocí algoritmu Conflict-based search (CBS), který zaručí, že nalezené řešení bude pro danou oblast optimální. Přechod agentů mezi jednotlivými sektory je řešen pomocí rezervačního mechanismu. Algoritmus se dále zabývá problémem selhání komunikace agenta s plánovačem oblasti.

3. Optimální řešení MAPF

Jak již bylo zmíněno v předchozí kapitole, problém MAPD je přirozeným rozšířením MAPF problému. Při řešení MAPD se nám tedy bude často hodit využít existující algoritmy na řešení MAPF, které si v této kapitole představíme.

Multi-agent path finding (MAPF) je problém definovaný grafem $G = (V,E)$ a množinou agentů $A = (a_1,a_2,\ldots,a_m)$. Každý agent a_i je určen počáteční pozicí $s_i \in V$ a cílovou pozicí $g_i \in V$. Agenti se v diskrétních časových krocích pohybují po grafu G, přičemž během jednoho kroku se agent buď přemístí na libovolné sousední políčko, nebo zůstane na své aktuální pozici. Cílem MAPF je najít pro každého agenta a_i cestu z s_i do g_i tak, aby spolu žádné dvě nalezené cesty nekolidovaly. Za kolizi se považuje *kolize ve vrcholu* a *kolize na hraně* definované v kapitole 1.1.

MAPF algoritmy se snaží minimalizovat nějakou cenovou funkci. Mezi dvě nejčastěji používané cenové funkce patří *makespan* a *sum of costs* (Stern a kol., 2019). *Makespan* vyjadřuje čas, jaký je potřeba k tomu, aby všichni agenti dorazili do svých cílových pozic. *Sum of costs* značí součet časů, které jednotliví agenti potřebují k tomu, aby dorazili na svou cílovou pozici. Nalezení optimálního řešení MAPF problému minimalizujícího jednu z výše uvedených cenových funkcí je NP-těžký problém (Surynek, 2010; Yu a LaValle, 2013). Stavový prostor problému totiž roste exponenciálně vzhledem k velikosti m (počtu agentů). Proto bylo postupně navrženo mnoho suboptimálních algoritmů, které umožňují řešit MAPF problém s velkým množstvím agentů. Nás však budou zajímat především optimální algoritmy.

Optimální MAPF algoritmy můžeme rozdělit do dvou kategorií - algoritmy založené na prohledávání (*search-based* v originále) a algoritmy založené na redukci (*reduction-based* v originále).

Algoritmy založené na prohledávání uvažují všechna možná rozmístění m agentů na V vrcholů. Mnohé algoritmy jsou založené na algoritmu A*. Příkladem je algoritmus M* (Wagner a Choset, 2011) nebo algoritmus A* vylepšený technikami *independence detection* a *operator decomposition* (Standley, 2010). Jiným přístupem jsou dvojúrovňové algoritmy, které na nižší úrovni umisťují agenty na vrcholy podle omezení určených na vyšší úrovni. Mezi dvojúrovňové algoritmy patří algoritmus *increasing cost tree search* (Sharon a kol., 2013) nebo algoritmus *conflict-based search* CBS či jeho vylepšená verze *improved conflict-based search* (Boyarski a kol., 2015). Algoritmy založené na prohledávání se typicky zaměřují na minimalizaci *sum of costs*, ale většinu z nich je možné snadno upravit na minimalizaci *makespanu*.

Algoritmy založené na redukci převádějí MAPF problém na jiné všeobecně známé problémy, jako je problém splnitelnosti Booleovských formulí (Surynek, 2014; Surynek a kol., 2016), problém s omezujícími podmínkami (Ryan, 2010), celočíselné lineární programování (Yu a LaValle, 2016) a mnoho dalších. Redukční algoritmy jsou většinou navrženy pro minimalizaci *makespanu*. Na rozdíl od algoritmů založených na prohledávání, změna cenové funkce na minimalizaci *sum of costs* často vyžaduje nový způsob redukce.

Následuje podrobnější popis dvou významných MAPF algoritmů, se kterými budeme v této práci dále pracovat.

3.1 CBS

Algoritmus Conflict-based search (CBS) (Sharon a kol., 2015) je jedním z algoritmů založených na prohledávání. Na rozdíl od algoritmů založených na A*, které vycházejí z prohledávání stavového prostoru, CBS prozkoumává konflikty mezi agenty.

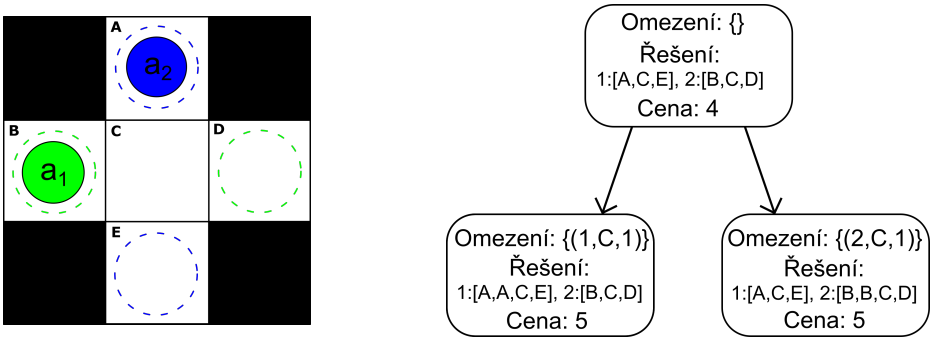
CBS řeší problém MAPF pomocí dekompozice problému na jednodušší problémy, a to problémy hledání nejkratší cesty pro jednoho agenta. V průběhu algoritmu jsou na jednotlivé agenty kladeny dodatečné podmínky, tzv. omezení. Omezení jsou ve tvaru (a,v,t) a říkají, že se agent a nesmí v čase t vyskytovat na pozici v. Princip algoritmu lze popsat následovně: Na počátku žádná dodatečná omezení neexistují. Algoritmus pro každého agenta nalezne nejkratší cestu, přičemž bere do úvahy všechna omezení, která jsou na daného agenta kladena. Pokud spolu žádné dvě nalezené cesty nekolidují, bylo nalezeno optimální bezkolizní řešení. Jinak uváží libovolnou kolizi dvou agentů (nechť se například v čase t na pozici x nachází agent a i agent b). Následně provede dvě rekurzivní volání algoritmu. V prvním voláním přidá dodatečné omezení pro agenta a, které mu zakáže se v čase t nacházet na pozici x. V druhém volání přidá obdobné omezení pro agenta b.

CBS je dvojúrovňový algoritmus. Na vyšší úrovni se vyhledávají konflikty v řešení a přidávají se nová omezení. Konflikt je čtveřice (a,b,v,t), která značí, že se agent a i agent b nacházejí v čase t na pozici v. Na nižší úrovni se pak uskutečňuje samotné hledání nejkratší cesty pro agenta se zadanými omezeními. Vyšší úroveň pracuje se stromem omezení. *Strom omezení* je binární strom, jehož vrcholy obsahují následující tři položky:

1. Množinu omezení tvaru (a_i,v,t).
2. Množinu cest. Tato množina obsahuje nejkratší cestu pro každého agenta, která respektuje omezení, jež jsou na daného agenta kladena. Nejkratší cesty jsou nalezeny na nižší úrovni algoritmu.
3. Celkovou cenu řešení, přičemž cenu řešení dostaneme jako součet cen jednotlivých cest.

Kořen stromu obsahuje prázdnou množinu omezení, neboť na počátku algoritmu žádná dodatečná omezení nejsou. Potomek vrcholu vždy dědí všechna omezení svého rodiče a přidá nové omezení pro jednoho z agentů. V rámci vyšší úrovně algoritmu se strom omezení prochází a hledá se v něm list L s nejmenší cenou řešení (v případě shody se preferuje list produkující menší množství konfliktů). Pro tento list se zkontroluje, zda je jeho množina cest bezkolizní. Pokud ano, množina cest představuje validní a optimální řešení problému. V opačném případě se vybere libovolný konflikt (a_i,a_j,v,t), který množina cest vytváří, a ten se vyřeší rozštěpením listu L na dva potomky. Levý potomek vyřeší konflikt přidáním omezení (a_i,v,t), pravý potomek přidáním omezení (a_j,v,t). Následně se pro oba nově vytvořené vrcholy spustí nižší úroveň algoritmu a nalezne se nejkratší cesta pro agenta s nově přidaným omezením.

Nižší úroveň algoritmu dostane na vstupu agenta a k němu přiřazená omezení a jeho úkolem je pro daného agenta a_i najít *nejkratší cestu* z s_i do g_i respektující zadaná omezení. Při hledání nejkratší cesty jsou ostatní agenti ignorováni. Autoři



(a) MAPF instance

(b) Strom omezení

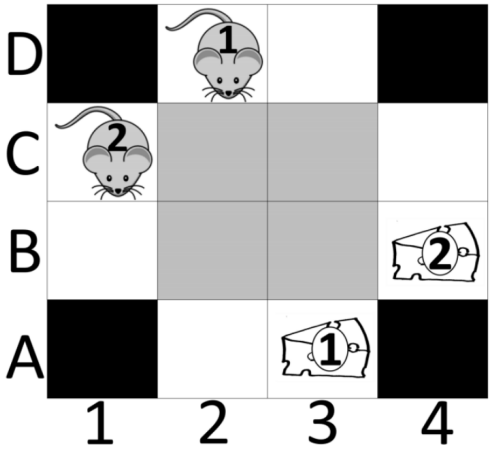
Obrázek 3.1: Příklad algoritmu CBS na konkrétní MAPF instanci

článku ke hledání cesty použili algoritmus A*, ale je možné použít libovolný jiný algoritmus na hledání nejkratší cesty.

Při dělení problému na podproblémy nejsou vytvořené podproblémy plně disjunktní - některá řešení jsou pro oba podproblémy stejná. Díky tomu dochází zbytečně k duplicitnímu řešení jednoho problému, a tím i ke snížení celkové efektivity algoritmu. Proto Li a kol. (2019) ve svém článku zavedli tzv. *pozitivní omezení*, které naopak vynutí přítomnost agenta v daném vrcholu v daném čase. Štěpíme-li na vyšší úrovni algoritmu list obsahující konflikt (a_i,a_j,v,t), levý potomek bude obsahovat standardní omezení (a_i,v,t), avšak pravý potomek bude obsahovat pozitivní omezení (a_i,v,t), které naopak přítomnost agenta a_i ve vrcholu v v čase t vynutí.

Příklad použití algoritmu je znázorněn na obrázku 3.1. Obrázek 3.1a zobrazuje instanci MAPF obsahující dva agenty. První agent se musí ze své aktuální pozice A dopravit na cílovou pozici E, druhý agent má za úkol přejet z pozice B na pozici D. Oba agenti tedy musí projet skrze pozici C. Algoritmus nejprve nalezne nejkratší cestu pro každého agenta. Následně se při první validaci zjistí, že spolu nejkratší cesty kolidují v čase 1 na pozici C. Strom omezení se tedy rozšíří o dva nové vrcholy. Jeden z nově vytvořených vrcholů bude obsahovat omezení $(1,C,1)$, druhý bude mít omezení $(2,C,1)$. Algoritmus postupně v obou nově vytvořených vrcholech přepočítá nejkratší cestu pro agenta, kterému bylo přidáno nové omezení. Následuje hledání listu s nejmenší cenou řešení. Protože mají všechny listy stromu omezení stejnou cenu řešení a produkují stejný počet konfliktů, algoritmus si vybere libovolný z nich. Během druhé validace se zjistí, že vybraný list obsahuje validní řešení, čímž běh algoritmu končí. Strom omezení, se kterým algoritmus pracoval, je zachycen na obrázku 3.1b.

Algoritmus CBS je exponenciální v počtu konfliktů, které se v průběhu výpočtu vyskytnou. Výsledky experimentů (Sharon a kol., 2015) ukazují, že algoritmus funguje dobře na takových instancích MAPF, ve kterých dominují koridory a úzká místa. Naopak na instancích MAPF s velkými otevřenými prostory funguje špatně. Příklad instance, na které CBS funguje špatně, je znázorněn na obrázku 3.2. Pro každého agenta existují 4 různé optimální cesty. Zároveň ale každá z 16 kombinací optimálních cest způsobí konflikt na jednom z šedých políček. Algoritmus CBS vyzkouší všechny kombinace optimálních cest předtím, než cestu jednoho z agentů prodlouží.



Obrázek 3.2: Příklad vstupu, na kterém CBS funguje špatně (Sharon a kol., 2015).

3.2 Redukce na SAT

Základní myšlenou všech algoritmů založených na redukci je vyjádřit problém pomocí podmínek, které musí každé validní řešení problému splňovat. Vyjádřením problému vznikne tzv. *deklarativní model*. Tento model je následně předán specializovanému řešiči, který se pro něj pokusí nalézt splňující ohodnocení. Deklarativní modely pro řešení MAPF jsou vytvářeny podle přístupu *planning as satisfiability* (Kautz a Selman, 1992). Tento přístup definuje MAPF problém jako množinu axiomů, pro které platí, že každý model splňující axiomy je validním řešením problému. Některé axiomy popisují počáteční a koncový stav, jiné zajišťují validitu prováděných akcí.

Deklarativní model vždy pracuje s pevně danou maximální délkou plánu. Redukční algoritmy proto typicky řeší MAPF problém postupnou iterací možných maximálních délek plánu, dokud není model pro nějakou délku plánu splnitelný. Dolní mez na maximální délku se často odhaduje pomocí jednoduché heuristiky, kdy se pro každého agenta najde nejkratší cesta ze startu do cíle a následně se dolní mez nastaví na maximum z nalezených cest.

Surynek (2012) zakódoval MAPF problém pomocí boolovských proměnných, které vyjadřují, zda se agent v daném čase nachází v určitém vrcholu, a pomocí podmínek zajišťujících validní přechody agentů mezi vrcholy. Surynek a kol. (2016) představili první deklarativní model MAPF minimalizující *sum of costs*. Kromě proměnných pro agenta nacházejícího se ve vrcholu zavedli také proměnné pro přechod agenta po hraně. Barták a Švancara (2019) popsali vylepšení modelu pomocí redukce množství proměnných.

V následujících sekcích představíme deklarativní model minimalizující *makespan* a jeho úpravu na model minimalizující *sum of costs*, který ve svém článku popsali Barták a Švancara (2019).

3.2.1 Model minimalizující Makespan

Jak již bylo zmíněno, deklarativní model se vždy definuje pro pevně danou délku plánu T. Barták a Švancara (2019) pracovali s modelem MAPF problému, který obsahuje dva typy proměnných a 7 podmínek. Pro každého agenta $a_i \in A$,

vrchol $v \in V$ a jednotku času $t \in \{0,\ldots,T\}$ zavedli proměnnou $At(v,i,t)$, která říká, jestli se agent a_i nachází v čase t ve vrcholu v. Dále pro každou hranu $(u,v) \in E$, agenta $a_i \in A$ a jednotku času $t \in \{0,\ldots,T-1\}$ vytvořili proměnnou $Pass(u,v,i,t)$, která určuje, jestli se agent a_i v čase t přesouvá po hraně z vrcholu $u \in V$ do vrcholu $v \in V$. Aby mohla proměnná $Pass(u,u,i,t)$ zachycovat situaci, ve které agent a_i zůstává v čase t ve vrcholu u, musí se graf G rozšířit o pomocné hrany (u,u) pro každé $u \in V$. Nakonec zavedli následující podmínky:

$$\forall a_i \in A : At(s_i,i,0) = 1 \tag{3.1}$$

$$\forall a_i \in A : At(g_i,i,T) = 1 \tag{3.2}$$

$$\forall a_i \in A, \forall t \in \{0,\ldots,T\} : \sum_{v \in V} At(v,i,t) \leq 1 \tag{3.3}$$

$$\forall v \in V, \forall t \in \{0,\ldots,T\} : \sum_{a_i \in A} At(v,i,t) \leq 1 \tag{3.4}$$

$$\forall u \in V, \forall a_i \in A, \forall t \in \{0,\ldots,T-1\} : At(u,i,t) \implies \sum_{(u,v) \in E} Pass(u,v,i,t) = 1 \tag{3.5}$$

$$\forall (u,v) \in E, \forall a_i \in A, \forall t \in \{0,\ldots,T-1\} : Pass(u,v,i,t) \implies At(v,i,t+1) \tag{3.6}$$

$$\forall (u,v) \in E, u \neq v, \forall t \in \{0,\ldots,T-1\} : \sum_{a_i \in A} (Pass(u,v,i,t) + Pass(v,u,i,t)) \leq 1 \tag{3.7}$$

Podmínky 3.1 a 3.2 zajistí, že každý agent bude začínat na své startovní a končit ve své cílové pozici. Podmínky 3.3 a 3.4 zaručí, že se žádný agent nebude vyskytovat ve více vrcholech najednou a že se v žádném vrcholu nebude nacházet více agentů současně. Nakonec podmínky 3.5, 3.6 a 3.7 vynutí správný pohyb agentů po grafu. Agent musí odejít z vrcholu po jedné z hran vedoucí z daného vrcholu (3.5), pokud použije nějakou hranu, musí dojít do vrcholu, ve kterém hrana končí (3.6). Nakonec podmínka 3.7 zabraňuje vzniku konfliktu na hraně.

Optimální hodnota *makespanu* je nalezena iterativně. Délka plánu T se postupně zvětšuje, dokud není nalezeno splňující ohodnocení modelu. Takto nalezené řešení je optimální řešení minimalizující *makespan*, neboť neexistuje žádné řešení s menším *makespanem*.

Ze způsobu zavedení proměnných je vidět, že jejich počet závisí na velikosti grafu, počtu agentů a délce plánu, ale není závislý na počtu konfliktů mezi agenty. Proto algoritmus může fungovat dobře i na instancích s velkým množstvím konfliktů.

Programovací jazyk Picat

Pro implementaci deklarativních modelů MAPF se často používá programovací jazyk Picat (Barták a kol., 2017; Atzmon a kol., 2018; Barták a Švancara, 2019). Tento jazyk totiž umožňuje jednoduchý a čitelný zápis podmínek, ze kterých je následně automaticky vytvořena výroková formule.

Picat je deklarativní jazyk, který je podobný Prologu. Kombinuje přístupy logického a funkcionálního programování s přístupy imperativního programování a skriptování. Picat poskytuje moduly, které umožňují převést problém popsaný v Picatu na Boolean satisfiability problem (Problém splnitelnosti booleovské formule) (SAT), Constraint programming (Programování s omezujícími podmínkami) (CP) nebo Mixed-Integer programming (Smíšené lineární programování) (MIP). Jazyku Picat se detailněji ve své knize věnovali Zhou a kol. (2015).

Implementace modelu v Picatu

Barták a Švancara (2019) ve svém článku představili implementaci deklarativního modelu v jazyce Picat, který optimálně řeší MAPF problém minimalizující *makespan*. Proměnné B jazyka Picat reprezentují proměnné pro jednotlivé vrcholy, proměnné C představují proměnné pro hrany. Proměnná N značí počet vrcholů grafu, E označuje počet hran grafu, As je seznam agentů, K označuje počet agentů a M reprezentuje velikost *makespanu*.

```
import sat.

path_for_delta(N,E,As,K,M) =>
    ME = M - 1,

    B = new_array(M,K,N),
    C = new_array(ME,K,E),

    % Podmínky 3.1 a 3.2: inicializace počátečního a koncového stavu.
    foreach(A in 1..K)
        (V,FV) = As[A],
        B[1,A,V] = 1,
        B[M,A,FV] = 1,
    end,

    B :: 0..1,
    C :: 0..1,

    % Podmínka 3.3: žádný agent se nebude nacházet ve více vrcholech najednou.
    foreach (T in 1..M, A in 1..K)
        sum([B[T,A,V] : V in 1..N]) #=< 1
    end,

    % Podmínka 3.4: v žádném vrcholu se nebude nacházet více agentů současně.
    foreach(T in 1..M, V in 1..N)
        sum([B[T,A,V] : A in 1..K]) #=< 1
    end,

    % Podmínka 3.7: pokud je hrana použita v jednom směru, nemůže být ve stejný
    % čas použita v opačném směru - zabraňuje vzniku konfliktu na hraně.
    foreach(T in 1..ME, EID in 1..E)
        oposit_edges(EID, E, EList)
```

```
    sum([C[T,A,W] : A in 1..K, W in EList]) #=< 1
end,

% Podmínka 3.5: pokud se agent nachází ve vrcholu, musí agent z daného
% vrcholu odejít po nějaké hraně z něj vedoucí.
foreach(T in 1..ME, A in 1..K, V in 1..N)
    out_edges(V,EList),
    B[T,A,V] #=> sum([C[T,A,W] : W in EList]) #= 1
end,

% Podmínka 3.6: pokud agent použije hranu, musí v následujícím časovém kroku
% dojít do vrcholu, do kterého daná hrana vede.
foreach(T in 1..ME, A in 1..K, EID in 1..E)
    edgeid(EID,_,V),
    C[T,A,EID] #=> B[T+1,A,V] #= 1
end,

solve(B).
```

4. Algoritmus/Řešení MAPD problému

V této kapitole představíme hierarchický algoritmus na řešení MAPD problémů. Algoritmus je založen na principu, který ve svém článku popsali Liu a kol. (2019b). V článku je princip hierarchického plánování popsaný relativně abstraktně, nastavuje poměrně striktní kritéria na rozdělení prostředí skladu na sektory a důraz je kladen především na řešení selhání komunikace mezi agenty a systémem.

Algoritmus, který popíšeme, bude parametrický ve smyslu využití dalších algoritmů během vlastního výpočtu. Pro hledání bezkolizních cest pro množinu agentů se budou používat upravené MAPF algoritmy, hierarchický algoritmus tedy můžeme parametrizovat právě pomocí použitého MAPF algoritmu. Stejně jako ve článku Liu a kol. (2019b) vyzkoušíme do výpočtu zapojit dynamické generování heat-mapy, která představuje způsob popisu dopravní situace v jednotlivých sektorech ve skladu. Heat-mapu budeme ovšem používat jiným způsobem.

4.1 Rozdělení prostředí

Abychom mohli použít na řešení problému hierarchický přístup, potřebujeme rozdělit prostředí skladu na několik sektorů. Rozdělení skladu musí splňovat jedinou podmínku - v rámci jednoho sektoru musí existovat cesta mezi libovolnými dvěma sousedními sektory. Sektor u je sousedem sektoru v, pokud je možné přejet přímo z u do v. Tato podmínka nám umožní se na sektor dívat jako na jeden celek a popsat prostředí skladu jako graf, jehož vrcholy představují jednotlivé sektory.

Přejíždět mezi sektory bude možné pouze přes speciálně označená místa, tzv. *přejezdy*. Přejezdem ze sektoru u do sektoru v budeme myslet dvojici sousedních políček x a y ve skladu takových, že x se nachází v sektoru u a y v sektoru v. Z definice sektoru mimo jiné vyplývá, že každý přejezd je jednosměrný. Budeme proto vyžadovat, aby mezi každými dvěma sousedními sektory existovaly alespoň dva různé *přejezdy*, každý pro jeden směr přejezdu. Přejezdů mezi dvěma sektory může samozřejmě existovat i více, potom by ale měly být rozmístěny rovnoměrně mezi všechny sousední sektory daného sektoru.

Hierarchický algoritmus popsaný v této kapitole bude fungovat na libovolném rozdělení skladu, které splňuje výše popsané podmínky.

4.2 Algoritmus

Plánování probíhá ve dvou fázích - nejprve se plánují cesty na globální úrovni, následně cesty na lokální úrovni. Fáze plánování se postupně střídají, dokud nejsou dokončeny všechny zadané úkoly.

Na globální úrovni jsou nezadané úkoly přiřazeny volným agentům. Úkoly jsou zadávány podle času zveřejnění, od nejstarších po nejmladší, ale samozřejmě by bylo možné pro přiřazování úkolů používat i jiné kritérium (např. uživatelem definovaná důležitost úkolů). Během jednoho plánování může být zadáno nejvýše

tolik úkolů, kolik v dané chvíli existuje volných agentů (neboť každý agent může mít v jednu chvíli zadaný maximálně jeden úkol). Poté je pro každého agenta, kterému byl nově zadán nějaký úkol, naplánovaná cesta. Výsledkem plánování pro jednoho agenta je posloupnost oblastí, přes které cesta daného agenta prochází. Cesta na globální úrovni je pro každého agenta naplánována pouze jednou, a to ve chvíli, kdy byl danému agentovi zadán nový úkol. Následně se agent pohybuje podle naplánované globální cesty, dokud nedokončí zadaný úkol.

Jakmile je plánování na globální úrovni dokončeno, započne druhá fáze plánování. V druhé fázi dochází paralelně k plánování cest na lokální úrovni v každém sektoru. Cílem je najít bezkolizní cesty pro všechny agenty v daném sektoru. Při plánování v jednom sektoru se agenti z jiných sektorů neuvažují. V momentě, kdy všechny sektory dokončí plánování na lokální úrovni, se všichni agenti posunou o jeden krok podél naplánované cesty. Nakonec je třeba vyřešit přesuny agentů mezi sousedními sektory. Poté následuje opět první fáze plánování.

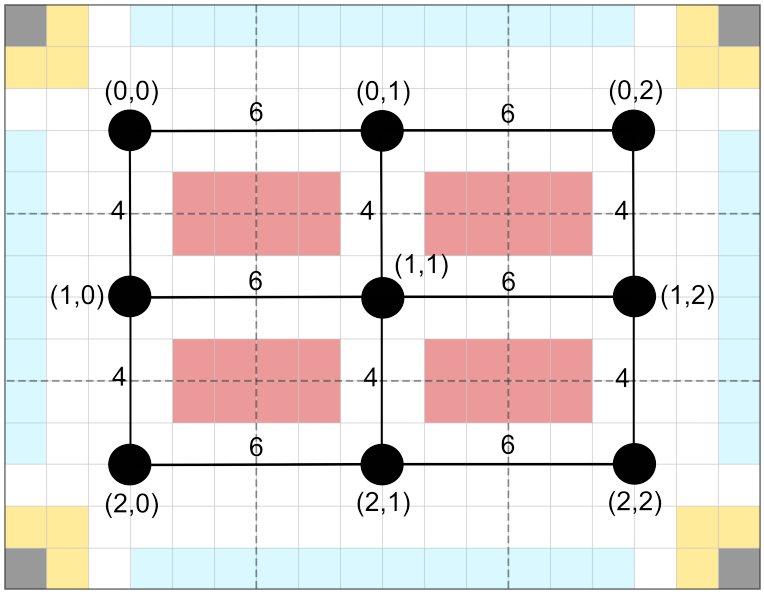
Algoritmus 1 obsahuje formálnější popis pseudokódu hierarchického algoritmu. Řádky 4-6 reprezentují plánování na globální úrovni, řádky 7-12 představují plánování na lokální úrovni. Protože se situace ve skladu mění prakticky každou časovou jednotku – volným agentům jsou přiřazovány nové úkoly, agenti přecházejí mezi sektory a dokončují zadané úkoly – budeme každou časovou jednotku pouštět obě dvě fáze plánování. While smyčka algoritmu se tedy bude vyhodnocovat každou časovou jednotku.

Z teoretického hlediska ale není přeplánovávání každou časovou jednotku nezbytné. Úplně stačí přeplánovávat v momentě, kdy dojde k nějaké *významné události*. Za významnou událost na globální úrovni bereme přidání nového úkolu, na lokální úrovni pak přejezd agenta mezi sektory nebo dokončení nějakého úkolu. Ještě sofistikovanější řešení by pouštělo první fázi plánování v případě výskytu významné události na globální úrovni a druhou fázi při výskytu významné události na lokální úrovni a to navíc pouze v sektorech, kterých se daná událost týká. V ostatních sektorech by se agenti drželi dříve naplánovaných cest.

Algorithm 1: Hierarchický algoritmus

Input: Seznam počátečních pozic agentů, graf skladu, grafy sektorů
Output: Seznam plánů pro všechny agenty

1 $\tau = \{\}$
2 $seznamPlanu = [\ ]$
3 **while** *existuje nedokončený úkol* **do**
4 | Aktualizuj τ `// Přidej všechny nově zveřejněné úkoly`
5 | Přiřaď úkoly z τ volným agentům
6 | Naplánuj cesty na globální úrovni
7 | **foreach** *sektor* **do**
8 | | Naplánuj cesty pro agenty v sektoru
9 | | Aktualizuj $seznamPlanu$ pro agenty v sektoru
10 | **end**
11 | Posuň všechny agenty o jeden krok podle $seznamPlanu$
12 | Přesuň agenty mezi sektory
13 **end**
14 **return** $seznamPlanu$



Obrázek 4.1: Příklad grafu skladu

Jednotlivé fáze plánování jsou podrobněji popsány v následujících sekcích.

4.3 Plánování na globální úrovni

Na globální úrovni se pracuje s grafem skladu. Vrcholy grafu skladu reprezentují jednotlivé sektory. Pokud sektor u sousedí se sektorem v, potom v grafu skladu existuje hrana mezi vrcholy u a v. Váha hrany (u,v) je rovna vzdálenosti středu sektoru u od středu sektoru v. Ukázkový graf skladu společně s váhami hran je zobrazen na obrázku 4.1. Jednotlivé vrcholy grafu jsou označeny souřadnicemi sektorů, které reprezentují.

Plánování na globální úrovni má na starost centrální řešič. Tento řešič si drží přehled o pozicích všech volných agentů a spravuje množinu úkolů τ. Množina τ obsahuje všechny úkoly, které byly zveřejněny. Řešič postupně přiděluje úkoly volným agentům a plánuje pro ně cestu v grafu skladu. Každý úkol má být vykonán právě jednou. Jakmile je tedy úkol zadán agentovi a_i, nemusí se řešič tímto úkolem již dále zabývat, protože se zodpovědnost za dokončení úkolu přenese na agenta a_i. Centrální řešič si tedy bude místo množiny τ udržovat množinu τ^*, což je množina τ, ze které jsou odebrány všechny úkoly, které již byly nějakému agentovi zadány.

Na počátku první fáze jsou všechny nově zveřejněné úkoly přidány do množiny τ^*. Z této množiny jsou pak úkoly zadávány volným agentům. Zadávání úkolů se dělá hladově - úkol je vždy přidělen nejbližšímu volnému agentovi. Pro výpočet vzdálenosti agentů se používá *Manhattanská metrika*. Po zadání úkolu je daný úkol z množiny τ^* odebrán. Následně je každému nově zaměstnanému agentovi naplánována cesta přes jednotlivé sektory. Při plánování cesty se pro agenta v grafu skladu hledá cesta ze sektoru, kde se agent aktuálně nachází, přes sektor obsahující místo zveřejnění úkolu do sektoru, ve kterém se nachází místo doručení přiřazeného úkolu. Nalezená cesta v grafu skladu představuje sekvenci sektorů, které agent během vykonávání úkolu navštíví. Cesta na globální úrovni

se plánuje pouze pro agenty, jimž byl nově zadán nějaký úkol.

Cesta v grafu skladu se hledá pomocí algoritmu A* (Hart a kol., 1968). Při hledání cesty jsou totiž ostatní agenti ignorováni, a tedy můžeme použít standardní algoritmus pro hledání nejkratší cesty v grafu. Protože budeme pracovat s grafy skladu, ve kterých má každý vrchol nejvýše jednoho souseda v každém ze čtyř směrů (nahoru, doprava, dolů, doleva), bude algoritmus A* pro odhad vzdálenosti používat Manhattanskou metriku. Manhattanská vzdálenost sektorů $\boldsymbol{x}$,$\boldsymbol{y}$ se souřadnicemi (x_1,x_2) a (y_1,y_2) je definovaná následovně:

$$d(\boldsymbol{x},\boldsymbol{y}) = |x_1 - y_1| + |x_2 - y_2|.$$

4.3.1 Využití heat-mapy

V dlouhodobém horizontu nemusí být vždy ideální posílat agenta nejkratší cestou. Při plánování cest na globální úrovni můžeme brát v úvahu také zatíženost jednotlivých sektorů. Zatíženost sektorů budeme charakterizovat *hustotou dopravy*.

Hustota dopravy v jednom sektoru se spočítá jako poměr počtu agentů nacházejících se v daném sektoru, kteří vykonávají nějaký úkol, a počtu volných políček v sektoru. Volní agenti, kteří se v daném sektoru nacházejí, se berou jako překážka, a tedy snižují počet volných políček. Hustotu dopravy v sektoru i tedy můžeme vyjádřit jako

$$hustota(i) = \frac{m_i}{free(i) - f_i},$$

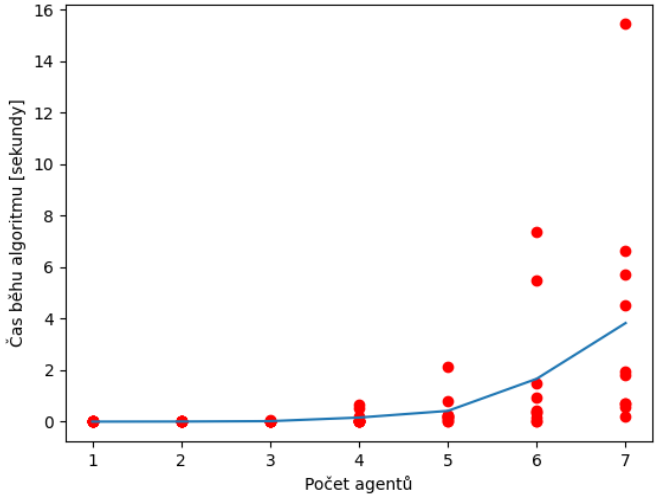
kde m_i značí počet agentů v sektoru i, kteří vykonávají nějaký úkol, f_i označuje počet volných agentů v sektoru i a $free(i)$ udává počet volných políček v sektoru i. Z definice hustoty dopravy je patrné, že bude nabývat hodnot v rozsahu (0,1). Hustota dopravy bude 0 pro sektory, ve kterých se nenacházejí žádní agenti vykonávající nějaký úkol, a 1 pro sektory, které mají všechna volná políčka obsazená nějakým agentem.

V první fázi algoritmu si z aktuálních informací o hustotě dopravy v jednotlivých sektorech můžeme vytvořit heat-mapu, která popisuje dopravní situaci ve skladu v daném čase. Dopravní situaci můžeme následně zohlednit při plánování cesty v grafu skladu. Můžeme se snažit posílat agenty do sektorů s nižší hustotou dopravy, čímž rozprostřeme agenty rovnoměrněji po celém skladu a snížíme množství dopravních zácp.

Hustota dopravy v sektoru j bude mít zřejmě vliv na váhu hrany v grafu skladu vedoucí do vrcholu j. Na hustotu dopravy se můžeme dívat jako na parametr, který ovlivňuje, jak dlouho bude trvat najít bezkolizní cesty pro všechny agenty v daném sektoru. Otázkou je, jak moc počet agentů v sektoru ovlivňuje dobu výpočtu algoritmu pro hledání bezkolizních cest.

Vztah hustoty dopravy a délky výpočtu

Vliv počtu agentů na délku výpočtu určíme experimentálně. Budeme měřit dobu běhu algoritmu CBS v jednom sektoru pro různý počet agentů. Pro fixní počet agentů si náhodně vygenerujeme startovní a cílové pozice. Vygenerované pozice nám společně s grafem sektoru definují instanci MAPF problému, kterou vyřešíme pomocí algoritmu CBS. Dobu běhu algoritmu budeme pro daný počet



Obrázek 4.2: Graf závislosti doby běhu algoritmu CBS na počtu agentů v jednom sektoru velikosti 6x5

agentů měřit na 10 různých MAPF instancích. Naměřené výsledky jsou zobrazeny v grafu 4.2. Červené tečky zachycují naměřené časy běhu pro jednotlivé MAPF instance. Modrá čára znázorňuje průměrnou dobu běhu algoritmu na 10 instancích.

Z grafu je vidět, že doba běhu algoritmu CBS v jednom sektoru roste exponenciálně vzhledem k počtu agentů. Váhu hrany tedy budeme chtít definovat tak, aby se při zvyšující se hustotě dopravy zvětšovala exponenciálně. Proto budeme váhu hrany z i do j definovat následovně:

$$w_{ij} = \frac{d_i + d_j}{2} + d_j^{const \times hustota(j)}, \tag{4.1}$$

kde d_i značí délku sektoru i, d_j označuje délku sektoru j a *const* reprezentuje kladný koeficient, který přenásobuje hustotu dopravy v sektoru j. První část sumy odpovídá původní váze hrany v grafu skladu – vzdálenosti středů sektorů. Hustota dopravy ovlivňuje druhou část sumy. Optimální hodnotu koeficientu *const* určíme experimentálně v kapitole 5.2.

4.4 Plánování na lokální úrovni

Na lokální úrovni se pracuje s grafy sektorů. Vrcholy grafu sektoru představují jednotlivá políčka v prostředí skladu. Ukázkový graf sektoru je znázorněn na obrázku 4.3.

Každý sektor má přiřazený svůj lokální řešič. Tento řešič si pamatuje pozice všech agentů v sektoru a stará se o plánování bezkolizních cest v rámci svého sektoru. Při plánování cest v jednom sektoru se agenti v jiných sektorech ignorují. Plánování jednotlivých sektorů jsou tedy na sobě navzájem nezávislá, takže mohou probíhat současně. Počet agentů v sektoru se průběžně mění – ze sousedních sektorů postupně přicházejí noví agenti a zároveň stávající agenti sektor opouštějí. Proto musí docházet k přeplánovávání cest každou časovou jednotku.

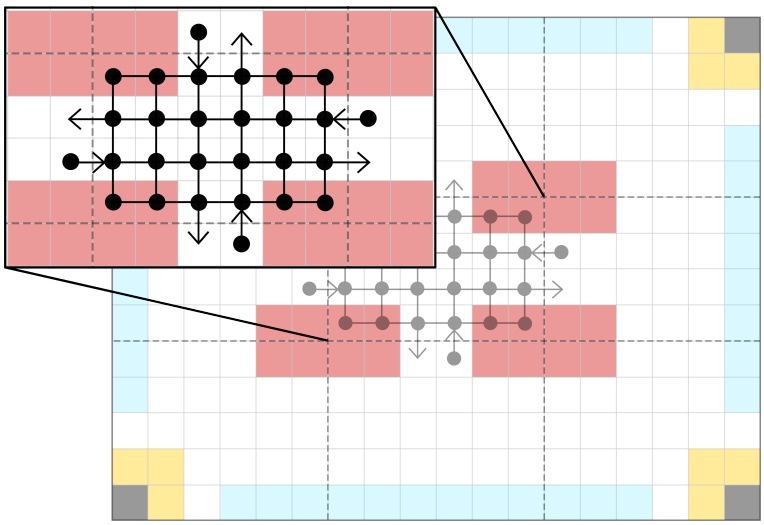
Na počátku druhé fáze si každý lokální řešič zformuluje MAPF problém odpovídající aktuální situaci v jeho sektoru. Způsob formulace MAPF problému bude detailněji popsán v následující sekci. Vytvořený MAPF problém je následně vyřešen pomocí jednoho z existujících MAPF algoritmů. Výsledkem lokálního plánování v jednom sektoru je množina bezkolizních cest pro všechny agenty v daném sektoru. Druhá fáze je zakončena přesunem agentů mezi sousedními sektory.

4.4.1 Přesuny agentů mezi sektory

Pokud budou v jednom časovém kroku přejíždět agent a_i z u do v a současně agent a_j z v do u, každý z nich bude muset použít jiný přejezd (kvůli jednosměrnosti přejezdů). Otázkou je, jak si budou sektory agenty mezi sebou předávat, aniž by došlo ke kolizi s ostatními agenty v daných sektorech.

Mějme agenta, který se nachází v sektoru u a který chce přejet do sousedního sektoru v. Aby mohl agent přejet mezi sektory, musí nejprve v sektoru u dojet na políčko přejezdu x. Jakmile na políčko x dorazí, sektor u předá agenta sektoru v a naplánuje agentovi cestu z x na y. Sektor v si následně na příští časový krok vyhradí políčko y pro přejíždějícího agenta. Vyhrazené políčko se pak při plánování cest v dalším kroku algoritmu považuje jako obsazené a přejíždějící agent se následujícího plánování cest vůbec neúčastní - cestu na příští krok má již naplánovanou.

Samotná implementace vyhrazování políček pro přejíždějící agenty využívá malého triku - ke každému sektoru jsou jednosměrnou hranou připojena políčka přejezdů ze sousedních sektorů. Tato uměle přidaná políčka slouží k jednoduššímu předávání agentů mezi sektory. Jakmile totiž nějaký agent dojede na políčko přejezdu z u do v v sektoru u, je předán sektoru v, které má uměle přidané políčko přejezdu z u. Sektor v si pak v následujícím kroku plánování sám zařídí přejezd agenta na vlastní území sektoru. Úspěšný přejezd agenta je vynucen tím, že z uměle přidaného políčka sektoru vede pouze jedna jednosměrná hrana, kterou je agent nucen v následujícím časovém kroku použít. Z toho důvodu se nemůže stát, že by agent zůstal stát na přejezdu v sektoru u.



Obrázek 4.3: Příklad grafu sektoru

4.4.2 Formulace MAPF problému

MAPF problém jednoho sektoru je definován následovně: množinu agentů A tvoří všichni agenti, kteří se nacházejí na území sektoru a vykonávají nějaký úkol. Startovní pozice agenta $a_i \in A$ je určena jeho aktuální pozicí. Cílová pozice agenta a_i, který má přiřazený úkol t_j, je určena následovně:

1. Pokud se v sektoru nachází místo vydání úkolu t_j, je cílová pozice místem vydání s_j,
2. pokud se v sektoru nachází místo doručení úkolu t_j, je cílová pozice místem doručení g_j,
3. jinak agent sektorem pouze projíždí a cílová pozice je tedy dána pozicí přejezdu do dalšího sektoru (dalším sektorem se myslí následující sektor podle naplánované globální cesty agenta).

Globální cesta každého agenta se tedy na lokální úrovni používá k určení cílové pozice daného agenta. Pokud existuje více různých přejezdů do sousedního sektoru, je agentovi jeden z přejezdů náhodně přidělen. Volní agenti, kteří se nacházejí na území sektoru, jsou považováni za překážku. Graf MAPF problému je definovaný grafem sektoru, ze kterého jsou odstraněny vrcholy, v nichž se nacházejí volní agenti.

Problém existence více přejezdů mezi sousedními sektory by bylo možné řešit také jiným způsobem. Výběr přejezdu bychom totiž mohli nechat na MAPF algoritmu. Algoritmu by byla místo jedné cílové pozice pro přejíždějící agenty zadána množina cílových pozic, která by představovala všechny možné přejezdy do sousedního sektoru. Konkrétní přejezdy by pak byly agentům přiřazeny až v rámci plánování bezkolizních cest.

Na MAPF problém vytvořený v rámci druhé fáze plánování v jednom sektoru není možné použít MAPF algoritmy přímo. Klasická definice MAPF problému počítá s tím, že jsou startovní a cílové pozice jednotlivých agentů navzájem různé. Pokud by totiž existovali dva agenti se stejnou startovní nebo cílovou pozicí, pak by problém nebyl řešitelný. V naší formulaci MAPF problému budou startovní pozice agentů navzájem různé vždy, neboť se žádní dva agenti nemohou nacházet v jednu chvíli na jednom políčku (startovní pozice je určena pozicí agenta v čase definování MAPF problému). Cílové pozice agentů ale budou často stejné, neboť pro všechny agenty projíždějící sektorem u do sektoru v bude jejich cílová pozice v sektoru u odpovídat pozici přejezdu do sektoru v. Proto budeme muset pracovat s verzí MAPF problému, která předpokládá, že každý agent po dosažení své cílové pozice zmizí.

Na lokální úrovni budeme používat algoritmy popsané v sekci 3.1 a 3.2, které upravíme tak, aby pracovaly s mizejícími agenty.

4.4.3 CBS

Úprava algoritmu CBS spočívá pouze v modifikaci hledání konfliktů. Agent, který dosáhl své cílové pozice v čase t, zmizí, a tedy již od času $t + 1$ nemůže být v konfliktu s žádným dalším agentem.

4.4.4 Redukce na SAT

Problém MAPF vyjádříme pomocí deklarativního modelu. Budeme vycházet z modelu popsaném v kapitole 3.2.1, který dále upravíme tak, aby vyhovoval našim potřebám.

Podmínky 3.1, 3.3, 3.4, 3.6 a 3.7 zůstanou zachovány beze změny. Podmínka 3.5 říká, že pokud se agent nachází v nějakém vrcholu, musí tento vrchol opustit po jedné z hran, které z daného vrcholu vedou. Tato podmínka platí pouze v případě, že se agent nenachází ve svém cílovém vrcholu. Pokud se agent nachází v cílovém vrcholu, zmizí a v žádném dalším vrcholu už se do konce řešení neobjeví. Podmínku 3.5 tedy nahradíme následujícími dvěma podmínkami:

$$\forall u \in V, \forall a_i \in A, \forall t \in \{0,\ldots,T-1\}, u \neq g_i : \\ At(u,i,t) \implies \sum_{(u,v)\in E} Pass(u,v,i,t) = 1 \tag{3.5a}$$

$$\forall u \in V, \forall a_i \in A, \forall t \in \{0,\ldots,T-1\}, u = g_i : \\ At(u,i,t) \implies \sum_{v\in V, t_1 \in \{t,\ldots,T\}} At(v,i,t_1) = 0 \tag{3.5b}$$

V klasickém MAPF problému se v čase rovném velikosti *makespanu* všichni agenti nacházejí na svých cílových pozicích. Tuto vlastnost problému zachycuje podmínka 3.2. Pro agenty, kteří po dosažení cílové pozice zmizí, neumíme předem říci, v jakém čase se budou na cílové pozici nacházet. Jisté ale je, že se na cílovou pozici v nějakém čase dostanou a následně zmizí. Místo podmínky 3.2 tedy zavedeme podmínku, která zajistí, že každý agent navštíví svou cílovou pozici právě jednou.

$$\forall a_i \in A : \sum_{t\in\{0,\ldots,T\}} At(g_i,i,t) \geq 1 \tag{3.2 nová}$$

Zdrojový kód v Picatu popisující upravené podmínky může vypadat následovně:

```
% Podmínka 3.2 nové: když agent dorazí do svého cílového vrcholu, zmizí.
foreach(A in 1..K)
    (V,FV) = As[A],
    if V !== 0 then
        sum([B[T,A,FV] : T in 1..M]) #= 1
    end
end,

% Podmínky 3.5a a 3.5b: pokud se agent nachází ve vrcholu (a tento vrchol není
% jeho cílovým vrcholem) musí agent z daného vrcholu odejít po nějaké hraně
% z něj vedoucí.
foreach(T in 1..ME, A in 1..K, V in 1..N)
    (_,FV) = As[A],
    if V !== FV then
        out_edges(V,EList),
        B[T,A,V] #=> sum([C[T,A,W] : W in EList]) #= 1
    else
        B[T,A,V] #=> sum([B[T1,A,V1] : T1 in T+1..M, V1 in 1..N]) #= 0
    end
end,
```

4.5 Problém s volnými agenty

Dosud jsme řešili pouze plánování pro agenty vykonávající nějaký úkol. Na globální úrovni se hledají cesty pro všechny agenty, kterým byl nově zadám nějaký úkol. Na lokální úrovni se pak plánují cesty pro všechny agenty, kteří vykonávají nějaký úkol, přičemž volné agenty při plánování považujeme za překážky. Pokud bychom nechali volné agenty stát na jednom místě, může se snadno stát, že MAPD problém nebude řešitelný. Volný agent by se totiž mohl nacházet na místě vyzvednutí nebo doručení nějakého nevykonaného úkolu, který by pak nebylo možné dokončit. Volným agentem se agent stane ihned po dokončení úkolu, což je ve chvíli, kdy se agent nachází na nějaké pozici z množiny koncových bodů úloh.

Liu a kol. (2019a) tento problém řešili pomocí mechanismu zvaného *reserving dummy paths*, který volné agenty posílá na nejbližší volnou pozici z množiny koncových bodů agentů. V této práci budeme také volné agenty posílat na nejbližší volný koncový bod agentů, avšak pouze v rámci sektoru, ve kterém se volný agent nachází. Pokud se volný agent nachází na nějakém koncovém bodě agentů, nikam se neposouvá. Pokud se již v sektoru žádný volný koncový bod agentů nenachází, volní agenti zůstávají stát na svém místě a nikam se nepohybují. Jak uvidíme v sekci 5.3.1, tento přístup sice nezaručí řešitelnost pro všechny MAPD problémy, ale funguje dobře pro instance s menším počtem agentů.

5. Experimenty

V následující kapitole otestujeme algoritmus popsaný v kapitole 4.2 v simulovaném prostředí. Nejprve experimentálně určíme nejvhodnější hodnotu parametru pro výpočet vah hran v grafu skladu. Dále budeme porovnávat běhy algoritmu pro různé kombinace použitých algoritmů na globální a lokální úrovni. V rámci posledního experimentu pak budeme zkoumat, jaký vliv na běh algoritmu má způsob rozdělení skladu na sektory. Spouštění experimentů je popsáno v příloze A.2.

5.1 Reprezentace prostředí

Pro účely experimentů byla vytvořena dvě simulovaná prostředí skladu. Prostředí byla vybudována tak, aby co nejvíce odpovídala reálnému prostředí automatizovaného skladu – ve středu skladu jsou pravidelně rozestavěné regály se zbožím, po obvodu skladu jsou rovnoměrně rozmístěny expediční oblasti. Vytvořená prostředí jsou znázorněna na obrázcích 5.1 a 5.2. Sklad je reprezentován čtverečkovanou mřížkou. Tento způsob reprezentace zachycuje skutečnost, že se agenti mohou ve skladu pohybovat ve čtyřech směrech – doprava, dolů, doleva a nahoru. Černá políčka představují neprůjezdné oblasti. Červená políčka reprezentují místa vydání (regály se zbožím), žlutá políčka místa doručení úkolů (expediční oblasti). Světle modrá políčka reprezentují možné počáteční pozice agentů (koncové body agentů). Vodorovné a svislé přerušované čáry znázorňují rozdělení skladu na sektory.

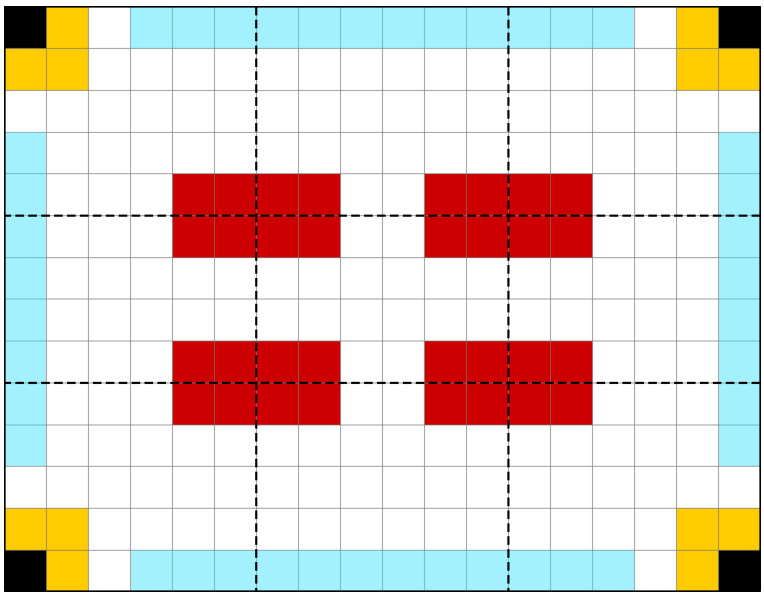
Simulovaná prostředí byla vytvořena v aplikaci online-MAPF (příloha A.3). Světle modré obdélníky a černé přerušované čáry do nich byly dokresleny dodatečně pro větší názornost. Obě prostředí splňují všechny podmínky z definice 7, a tedy jsou dobře formovaná.

Generování úkolů závisí na dvou parametrech - počtu úkolů a *frekvenci zveřejňování* úkolů. Frekvence zveřejňování úkolů značí počet úkolů, které jsou v každém časovém kroku zveřejněny. Frekvence rovná 1 znamená, že je v každém časovém kroku zveřejněn jeden úkol, frekvence 0,5 naopak značí, že je jeden úkol zveřejněn v každém druhém časovém kroku. Zadaný počet úkolů je vygenerován tak, že je náhodně vybráno místo vydání z množiny míst vydání a místo doručení z množiny míst doručení. Časy zveřejnění úkolů jsou následně zvoleny tak, aby odpovídaly zadané frekvenci zveřejňování. Počáteční pozice agentů jsou zvoleny náhodně z množiny počátečních pozic agentů.

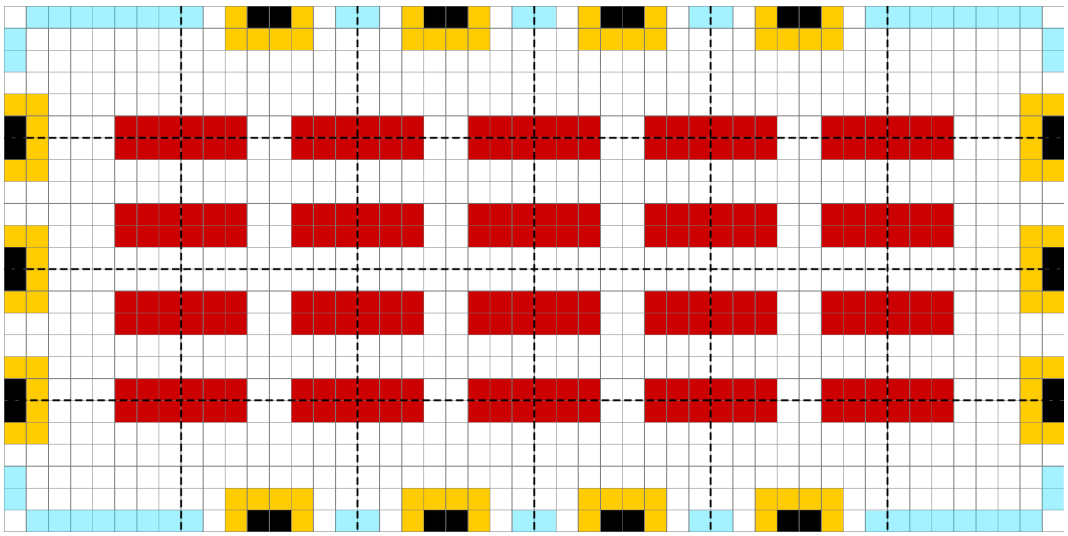
Pro účely experimentu tedy můžeme MAPD instanci popsat pomocí grafu prostředí, počtu agentů, počtu úkolů a frekvencí zveřejňování úkolů. Pro daný graf je následně vygenerováno zadané množství úkolů a agentů.

5.2 Určení koeficientu pro výpočet váhy hrany

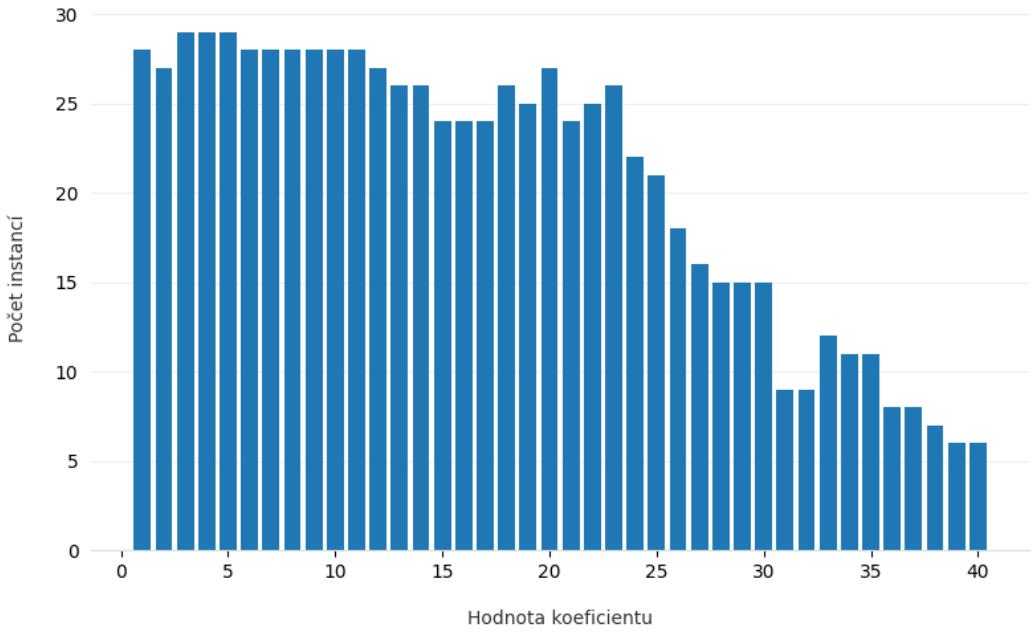
Optimální hodnotu koeficientu *const*, který se používá ve vzorečku 4.1, určíme experimentálně. Pro různé hodnoty *const* budeme postupně pouštět hierarchický algoritmus s algoritmem CBS na lokální úrovni a A* využívající heat-mapu



Obrázek 5.1: Prostředí skladu velikosti 18x14



Obrázek 5.2: Prostředí skladu velikosti 48x24



Obrázek 5.3: Graf závislosti počtu instancí s lepším makespanem řešení (než referenční řešení) na hodnotě koeficient *const*

s danou *const* na globální úrovni a budeme zaznamenávat *makespan* nalezeného řešení. Experiment budeme provádět na MAPD instance definované grafem prostředí 5.1 s 10 agenty a 50 úkoly s frekvencí zveřejňování 1. *Makespan* řešení budeme hledat pro 50 různých MAPD instancí. Jednotlivé instance se budou lišit v počátečních pozicích agentů a místech vydání a doručení úkolů. Graf prostředí, počet agentů i počet úkolů bude pro všechny instance stejný.

Experiment budeme realizovat pro hodnoty $const \in \{1, \ldots, 49\}$. Zároveň si také pro každou MAPD instanci změříme *makespan* řešení algoritmu, který na lokální úrovni heat-mapu nepoužívá. Pro každou hodnotu *const* nás pak bude zajímat, na kolika MAPD instancích měl algoritmus s danou *const* menší nebo stejný *makespan* oproti algoritmu, který heat-mapu nepoužíval.

Výsledky experimentu jsou znázorněny na obrázku 5.3. Z grafu je vidět, že pro hodnoty *const* od 1 do 14 algoritmus využívající heat-mapu dosahuje řešení s lepším nebo stejně dobrým *makespanem* než algoritmus bez heat-mapy, a to alespoň v 50 % případů (více než 25 instancí z 50). Nejlepších výsledků dosahuje algoritmus pro koeficient *const* s hodnotu 3, 4 nebo 5.

V dalších experimentech budeme používat koeficient *const* s hodnotou 5.

5.3 Porovnání využití různých algoritmů na globální a lokální úrovni

V následujícím experimentu budeme porovnávat *makespan* a runtime běhu hierarchického algoritmu při použití různých algoritmů na globální a lokální úrovni. Na globální úrovni budeme používat algoritmus A* a algoritmus A* využívající heat-mapu. Na lokální úrovni pak algoritmus CBS a redukci MAPF problému na SAT.

Experimenty budeme provádět na MAPD instanci definované grafem prostředí 5.1, tentokrát s 250 úkoly. Budeme zkoušet 6 různých frekvencí zveřejňování úkolů: 0.2, 0.5, 1, 2, 5 a 10. Pro každou frekvenci vyzkoušíme 5 různých počtů agentů: 5, 10, 15, 20 a 25. Pro každou dvojici parametrů (počet agentů a frekvence zveřejňování úkolů) si vygenerujeme 3 různé instance MAPD problému, na kterých spustíme hierarchický algoritmus popsaný v předchozí kapitole. Timeout bude nastaven na 600 sekund a po uplynutí tohoto času bude běh algoritmu prohlášen za neúspěšný. Tabulka 5.1 zobrazuje medián *makespanu* nalezených řešení (v krocích) 3 instancí se stejnou dvojicí parametrů a runtimu na jeden krok algoritmu (v sekundách). Symbol – v tabulce značí, že hierarchický algoritmus nedoběhnul pro žádnou ze tří MAPD instancí.

Makespan řešení

Z tabulky je vidět, že *makespan* řešení se zmenšuje s rostoucím množstvím agentů v systému. To je dáno tím, že při větším množství agentů musí každý agent vykonat menší množství úkolů. Nižší frekvence zadávání úkolů mívají *makespan* menší než vyšší frekvence, neboť počet úkolů je konstantní, a tedy je při nižších frekvencích potřeba více kroků k přidání všech úkolů do systému.

Runtime algoritmu

Runtime algoritmu se také zvyšuje s rostoucím počtem agentů v systému, neboť při zvětšujícím se počtu agentů se zároveň zvyšuje hustota dopravy v jednotlivých sektorech, a tím pádem také častěji dochází ke kolizím mezi agenty během hledání bezkolizních cest na lokální úrovni. Naopak růst frekvence zadávání úkolů runtime algoritmu prakticky neovlivňuje.

Nárůst runtimu algoritmu s CBS je znatelně větší než nárůst algoritmu s redukcí na SAT.

SAT vs. CBS

Hierarchický algoritmus používající CBS na globální úrovni nachází řešení s menším *makespanem* než algoritmy používají redukci na SAT. To bude nejspíše způsobeno tím, že algoritmus CBS používá jinou cenovou funkci než redukce na SAT. Použitý algoritmus CBS se snaží minimalizovat *sum of costs*, zatímco pro redukci na SAT jsme použili model minimalizující *makespan*. Minimalizovat *makespan* v jednom sektoru nemusí být z dlouhodobého hlediska optimální, neboť plánování cest v jednotlivých sektorech se navzájem neovlivňuje. Pokud je tedy nějakému agentovi naplánovaná zbytečně delší cesta přes jeden sektor (minimalizující celkový *makespan* v daném sektoru), nemá to žádný vliv na plánování jeho cesty skrz následující sektor. Agenti tedy cestou při plnění úkolu naberou větší zpoždění, než kdyby sektory při plánování cest minimalizovaly *sum of costs*.

Při nižším počtu agentů v systému bývá runtime algoritmu používající CBS až 10krát nižší než runtime algoritmu aplikující redukci na SAT. Se zvyšujícím se počtem agentů se rozdíl runtimů snižuje - algoritmus s CBS bývá v systému s 15-20 agenty pouze 2krát rychlejší než algoritmus s redukcí na SAT. To je dáno tím, že algoritmus CBS je exponenciální v počtu konfliktů mezi agenty. S vyšší hustotou agentů totiž dochází ke vzniku více konfliktů mezi agenty. Naopak při

frekv	agenti	CBS + naive mks	CBS + naive rtime	CBS + heat mks	CBS + heat rtime	SAT + naive mks	SAT + naive rtime	SAT + heat mks	SAT + heat rtime
0,2	5	1529	0,027	1541	0,028	1565	0,254	1593	0,256
	10	1272	0,035	1272	0,030	1272	0,289	1274	0,281
	15	–	–	–	–	–	–	–	–
	20	–	–	–	–	–	–	–	–
	25	–	–	–	–	–	–	–	–
0,5	5	1510	0,029	1508	0,029	1536	0,258	1544	0,254
	10	779	0,068	762	0,075	813	0,42	800	0,430
	15	530	0,124	536	0,192	568	0,530	563	0,537
	20	525	0,108	–	–	521	0,546	525	0,552
	25	–	–	–	–	–	–	–	–
1	5	1498	0,031	1486	0,030	1546	0,268	1548	0,268
	10	754	0,083	750	0,077	804	0,45	788	0,442
	15	507	0,141	507	0,161	553	0,556	566	0,554
	20	394	1,387	390	0,349	426	0,634	435	0,641
	25	321	0,361	314	0,384	370	0,702	–	–
2	5	1490	0,029	1489	0,029	1525	0,277	1543	0,277
	10	746	0,069	741	0,072	782	0,456	792	0,456
	15	510	0,174	509	0,244	552	0,571	554	0,578
	20	384	0,218	–	–	–	–	430	0,658
	25	318	0,614	–	–	–	–	–	–
5	5	1510	0,045	1513	0,048	1552	0,282	1562	0,291
	10	766	0,106	765	0,121	810	0,471	794	0,461
	15	508	0,328	501	0,237	558	0,556	555	0,537
	20	381	0,330	395	0,276	–	–	430	0,617
	25	308	0,370	306	0,540	353	0,711	–	–
10	5	1484	0,028	1487	0,030	1524	0,267	1562	0,262
	10	752	0,074	736	0,077	787	0,430	786	0,439
	15	511	0,174	517	0,202	549	0,582	541	0,556
	20	379	0,247	380	0,311	432	0,687	426	0,692
	25	–	–	–	–	–	–	–	–

Pozn: mks značí makespan (v krocích), rtime značí runtime jednoho kroku (v sekundách)

Tabulka 5.1: Výsledky hierarchického algoritmu pro různé kombinace algoritmů na globální a lokální úrovni

redukci na SAT se sice se zvyšujícím se počtem agentů zvětšuje deklarativní model (pro nové agenty jsou zavedeny nové proměnné), ale růst modelu je pouze polynomiální. Vzhledem ke snižování rozdílu mezi runtimu algoritmů CBS a redukci na SAT se můžeme domnívat, že kdybychom počet agentů dále zvyšovali, algoritmus s redukcí na SAT by co do runtimu algoritmus s CBS překonal. K ověření této hypotézy bychom ale museli vyřešit problém, že hierarchický algoritmus s vyšším počtem agentů často nedoběhne (viz sekce 5.3.1).

Klasický A* vs. A* s heat-mapou

Algoritmus A* využívající heat-mapu se snaží dopravu rozprostřít rovnoměrně mezi sektory. Tato snaha sice může způsobit zvětšení *makespanu* řešení (neboť mohou být někteří agenti posláni delší cestou na globální úrovni, a tedy přes více sektorů), ale zároveň to umožňuje snížit výskyt kolizí mezi agenty v jednom sektoru.

SAT v kombinaci s klasickým algoritmem A* dosahuje o trochu nižšího *makespanu* než SAT s algoritmem A* využívající heat-mapu. V případě algoritmu CBS nejde jednoznačně určit, jestli je algoritmus A* využívající heat-mapu lepší než klasický A*, záleží totiž na konkrétní MAPD instanci. V některých experimentech je *makespan* A* využívající heat-mapu menší než *makespan* klasického A*, v jiných je tomu naopak.

Runtime algoritmů s klasickým A* je podobný runtimu A* využívajícího heat-mapu. V případě CBS umí A* využívající heat-mapu snížit počet konfliktů mezi agenty v jednom sektoru, a tím značně snížit runtime – viz výsledek experimentu s 20 agenty a frekvencí 1.

Otázkou je, proč na některých instancích MAPD algoritmus nedoběhl.

5.3.1 Proč algoritmus někdy nedoběhne

Při experimentech může algoritmus skončit neúspěchem ze dvou důvodů - algoritmus počítal moc dlouho (vypršel timeout) nebo v nějakém sektoru nebyly nalezeny bezkolizní cesty pro agenty v daném sektoru. Timeout byl při experimentech nastaven na 600 sekund. Všechny experimenty, které nedoběhly, skončily neúspěchem z druhého důvodu. Proč ale pro některé MAPD instance nebyly nalezeny bezkolizní cesty? Odpověd souvisí s chováním volných agentů.

Jak bylo popsáno v sekci 4.5, volní agenti jsou v rámci svého sektoru posíláni na nejbližší volný koncový bod agentů. Pokud se v sektoru žádný takový bod nenachází, volní agenti zůstávají stát na svém místě. A právě toho chování je zdrojem potíží.

Před začátkem hierarchického algoritmu se všichni agenti nacházejí na koncových bodech agentů. V průběhu algoritmu jsou agentům zadávány úkoly, které do systému postupně přicházejí. Agenti se v sektorech obměňují, protože jsou posíláni napříč sektory v rámci plnění úkolů. V žádném sektoru se tedy nenachází více volných agentů, než je počet koncových bodů sektorů.

Problém nastává v momentě, kdy se v některém sektoru nachází více volných agentů, než je počet koncových bodů agentů. V tu chvíli totiž bude v daném sektoru určitě existovat agent, který zůstal stát na místě doručení svého posledního úkolu. A právě tento tzv. *blokující agent* může způsobit, že MAPF problém definovaný pro daný sektor ve druhé fázi algoritmu nebude řešitelný (pokud se bude

frekv	CBS + naive	CBS + heat	SAT + naive	SAT + heat
0,2	94	96	72	67
0,5	95	92	91	105
1	322	320	350	355
2	307	304	341	342
5	295	302	337	350

Tabulka 5.2: Průměrný počet kroků algoritmu končícího neúspěchem na MAPD instance s 25 agenty

místo doručení nějakého úkolu shodovat s pozicí blokujícího agenta). Blokující agenti se začnou typicky objevovat ke konci běhu algoritmu, kdy bude zbývat několik málo nedokončených úkolů. Všichni volní agenti pak totiž budou koncentrováni v sektorech s expedičními oblastmi (neboť v těchto sektorech dokončili svůj úkol).

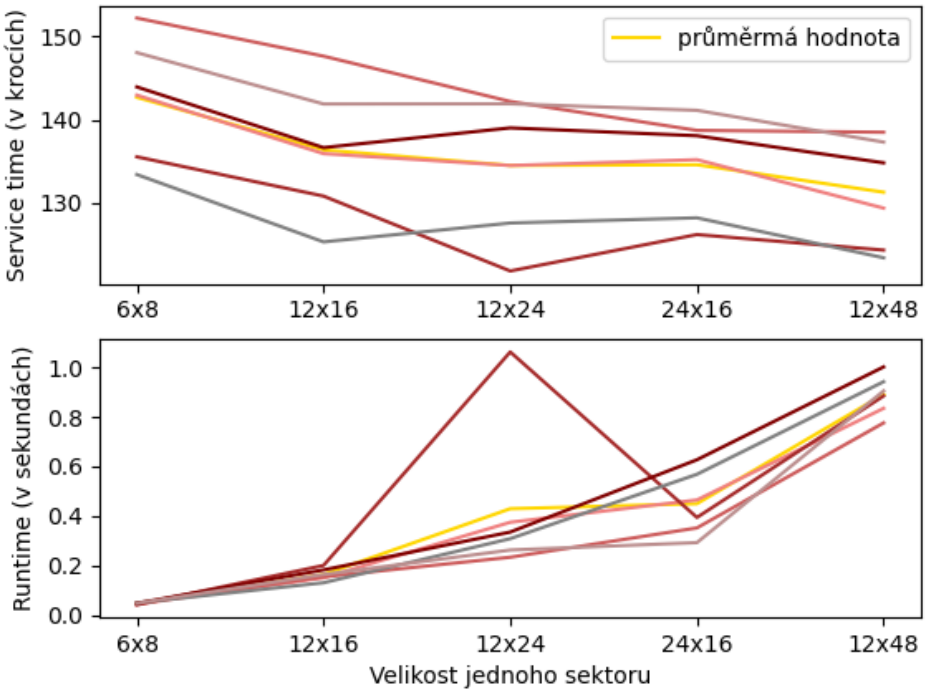
V prostředí 5.1 se nacházejí 4 sektory s expediční oblastí. Každý z těchto tzv. *expedičních sektorů* obsahuje přesně 5 koncových bodů agentů. Zároveň každá expediční oblast obsahuje 3 možná místa doručení úkolů. Při 20 nebo 25 agentech se budou blokující agenti objevovat ke konci běhu algoritmu prakticky vždy.

Tabulka 5.2 zachycuje, kolik kroků hierarchický algoritmus průměrně vykonal na MAPD instanci s 25 agenty, než skončil neúspěchem. Uvážíme-li, že algoritmus používající CBS s klasickým A* doběhnul na instanci s 25 agenty a frekvencí 2 s *makespanem* 318 kroků (viz tabulka 5.1), můžeme konstatovat, že pro frekvence zveřejňování úkolů větší než 1 se průměrný počet kroků neúspěšných běhů algoritmu blíží *makespanu* řešení úspěšných běhů algoritmu. Což přesně odpovídá domněnce, že se blokující agenti objevují v posledních krocích algoritmu. Oproti tomu instance s frekvencí menší než 1 končí neúspěchem během prvních 100 kroků algoritmu. To bude pravděpodobně způsobeno tím, že při malých frekvencích přicházejí úkoly do systému s většími časovými rozestupy (při frekvenci 0,2 je během prvních 100 kroků do systému přidáno pouze 20 úkolů). Malý počet úkolů pak znamená hodně volných agentů, a tedy i velkou šanci na výskyt blokujícího agenta.

5.4 Porovnání různých rozdělení skladu na sektory

Poslední experiment se bude zabývat tím, jaký vliv na chování algoritmu má způsob rozdělení skladu na oblasti. Budeme pracovat s hierarchickým algoritmem používajícím CBS na lokální úrovni a klasický A* na globální úrovni. Měření budeme provádět na MAPD instanci definované grafem prostředí 5.2. Počet agentů v systému, počet úkolů i frekvence zveřejňování úkolů budou fixní. Měnit se bude pouze počet (a velikost) sektorů, na které bude sklad rozdělen.

Budeme zkoušet 5 způsobů rozdělení skladu na sektory: 24 sektorů velikosti 6×8, 6 sektorů velikosti 12×16, 4 sektory velikosti 12×24, 3 sektory velikosti 24×16 a 2 sektory velikosti 12×48. Postupně vygenerujeme 10 různých MAPD instancí s 10 agenty, 50 úkoly a frekvencí zveřejňování 2. Pro každou MAPD instanci



Obrázek 5.4: Graf závislosti makespanu a runtimu na velikostech sektorů

vyzkoušíme všech 5 možných rozdělení na sektory. Výsledky experimentu jsou zobrazeny na obrázku 5.4. Každá čára vždy reprezentuje jednu MAPD instanci a znázorňuje, jak se mění *service time* řešení a runtime algoritmu při změně rozdělení mapy na sektory.

Service time řešení

Z grafu je vidět, že se zvětšující se velikostí sektoru se *service time* řešení zmenšuje. Platí, že *service time* řešení se sektory velikosti 6×8 je vždy větší než *service time* řešení s většími sektory. *Service time* řešení se ovšem nikdy neliší o více než 15 kroků.

Runtime algoritmu

Dále můžeme pozorovat, že při zvětšující se velikosti sektoru dochází k výraznému zvýšení velikosti runtimu na jeden krok algoritmu. Runtime algoritmu se sektory velikosti 12×48 byl skoro 10krát větší než runtime algoritmu se sektory 6×8. Rozdíl v runtimech algoritmu je způsoben tím, že při zvětšující se velikosti sektorů (a tedy zmenšujícím se počtu sektorů) se v sektorech zároveň zvyšuje hustota dopravy. Algoritmus CBS na lokální úrovni hledá bezkolizní cesty na větším území a pro větší počet agentů, mezi kterými pak dochází častěji ke konfliktům. Z pohledu doby běhu algoritmu se tedy hierarchický přístup vyplatí.

Závěr

V této práci jsme se zabývali problémem Multi-agent pickup and delivery (MAPD), což je problém hledání bezkolizních cest pro skupinu agentů v automatizovaném skladu. Problém jsme si nejprve formálně zadefinovali a podívali jsme se na způsoby řešení, které se v literatuře vyskytují. Offline přístup nachází řešení s nejmenším makespanem, ale vyžaduje, aby byla množina úkolů známa předem. Online přístup tímto omezením netrpí - na MAPD problém se dívá jako na posloupnost MAPF problémů, takže zvládá zpracovávat postupně zveřejňované úkoly. Oba předchozí přístupy se ale špatně škálují na větší MAPD instance, což je pro reálné použití nepraktické. A právě problém se škálovatelností se snaží vyřešit hierarchický přístup, který řeší problém rozdělením na menší podproblémy. Následně jsme představili hierarchický algoritmus na řešení MAPD, který pracuje s mapou skladu rozdělenou na oblasti a provádí plánování na dvou úrovních - globální a lokální. Na globální úrovni dochází k zadávání úkolů volným agentům a plánování cest skrze oblasti. Plánování na lokální úrovni hledá vlastní cesty v jednotlivých oblastech a probíhá v každé oblasti zvlášť. Cesty na lokální úrovni jsou hledány pomocí upravených MAPF algoritmů, konkrétně algoritmu CBS a redukce na SAT.

Experimentálně jsme zjistili, že hierarchický algoritmus používající algoritmus CBS dosahuje lepších výsledků než algoritmus využívající redukci na SAT. Algoritmus používající CBS má menší runtime než algoritmus s redukcí na SAT a zároveň nachází řešení s menším makespanem. V dalším experimentu jsme zkoumali vliv způsobu rozdělení mapy skladu na výkon algoritmu. Zjistili jsme, že zmenšováním velikostí oblastí se výrazně snižuje runtime algoritmu a zároveň se service time nalezeného řešení zvyšuje pouze minimálně, takže hierarchický přístup představuje dobrý kompromis mezi dobou výpočtu a kvalitou řešení.

V současné implementaci hierarchického algoritmu dochází k přeplánování každou jednotku času, což ale není z teoretického hlediska nutné. Sofistikovanější časování (podrobněji popsané v sekci 4.2) by mohlo přispět k celkovému zmenšení runtimu algoritmu. Implementace algoritmu by také mohla být vylepšena změněním způsobu zacházení s volnými agenty, který by zaručil řešitelnost všech dobře formovaných MAPD instancí. Navržený hierarchický algoritmus pracuje s ručně daným rozdělením skladem na oblasti. Automatizace dělení by proto mohla být dalším možným rozšířením.

Předmětem dalšího studia by také mohlo být použití dalších MAPF algoritmů na lokální úrovni, například redukce na SAT, která by jako cenovou funkci minimalizovala součet cen (tzv. *sum of costs*). Další možností by bylo využít kombinaci MAPF algoritmů - některé oblasti by používaly jeden MAPF algoritmus, ostatní by používaly jiný. Kombinování MAPF algoritmů by mohlo fungovat dobře ve skladech, které by byly na oblasti rozděleny nerovnoměrně.

Seznam použité literatury

Atzmon, D., Stern, R., Felner, A., Wagner, G., Barták, R. a Zhou, N. (2018). Robust multi-agent path finding. In André, E., Koenig, S., Dastani, M. a Sukthankar, G., editors, *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pages 1862–1864. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM.

Barták, R., Zhou, N., Stern, R., Boyarski, E. a Surynek, P. (2017). Modeling and solving the multi-agent pathfinding problem in picat. In *29th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2017, Boston, MA, USA, November 6-8, 2017*, pages 959–966. IEEE Computer Society. doi: 10.1109/ICTAI.2017.00147.

Barták, R. a Švancara, J. (2019). On sat-based approaches for multi-agent path finding with the sum-of-costs objective. In Surynek, P. a Yeoh, W., editors, *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019*, pages 10–17. AAAI Press.

Botea, A., Müller, M. a Schaeffer, J. (2004). Near optimal hierarchical path-finding. *J. Game Dev.*, **1**(1), 1–30.

Boyarski, E., Felner, A., Stern, R., Sharon, G., Betzalel, O., Tolpin, D. a Shimony, S. E. (2015). ICBS: the improved conflict-based search algorithm for multi-agent pathfinding. In Lelis, L. a Stern, R., editors, *Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015, 11-13 June 2015, Ein Gedi, the Dead Sea, Israel*, pages 223–225. AAAI Press.

Gelfond, M. a Lifschitz, V. (1988). The stable model semantics for logic programming. In Kowalski, R. A. a Bowen, K. A., editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes)*, pages 1070–1080. MIT Press.

Hart, P. E., Nilsson, N. J. a Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, **4**(2), 100–107. doi: 10.1109/TSSC.1968.300136.

Kautz, H. A. a Selman, B. (1992). Planning as satisfiability. In Neumann, B., editor, *10th European Conference on Artificial Intelligence, ECAI 92, Vienna, Austria, August 3-7, 1992. Proceedings*, pages 359–363. John Wiley and Sons.

Li, J., Harabor, D., Stuckey, P. J., Ma, H. a Koenig, S. (2019). Disjoint splitting for multi-agent path finding with conflict-based search. In Benton,

J., Lipovetzky, N., Onaindia, E., Smith, D. E. a Srivastava, S., editors, *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019*, pages 279–283. AAAI Press.

Li, J., Tinka, A., Kiesel, S., Durham, J. W., Kumar, T. K. S. a Koenig, S. (2020). Lifelong multi-agent path finding in large-scale warehouses. *CoRR*, **abs/2005.07371**.

Liu, M., Ma, H., Li, J. a Koenig, S. (2019a). Task and path planning for multi-agent pickup and delivery. In Elkind, E., Veloso, M., Agmon, N. a Taylor, M. E., editors, *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*, pages 1152–1160. International Foundation for Autonomous Agents and Multiagent Systems.

Liu, Z., Wang, H., Zhou, S., Shen, Y. a Liu, Y. (2019b). Coordinating large-scale robot networks with motion and communication uncertainties for logistics applications. *CoRR*, **abs/1904.01303**.

Ma, H., Li, J., Kumar, T. K. S. a Koenig, S. (2017). Lifelong multi-agent path finding for online pickup and delivery tasks. *CoRR*, **abs/1705.10868**.

Ma, H., Hönig, W., Kumar, T. K. S., Ayanian, N. a Koenig, S. (2018). Lifelong path planning with kinematic constraints for multi-agent pickup and delivery. *CoRR*, **abs/1812.06355**.

Nguyen, V., Obermeier, P., Son, T. C., Schaub, T. a Yeoh, W. (2017). Generalized target assignment and path finding using answer set programming. In Sierra, C., editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1216–1223. ijcai.org. doi: 10.24963/ijcai.2017/169.

Pianpak, P., Son, T. C., Toups, Z. O. a Yeoh, W. (2019). A distributed solver for multi-agent path finding problems. In *Proceedings of the First International Conference on Distributed Artificial Intelligence, DAI 2019, Beijing, China, October 13-15, 2019*, pages 2:1–2:7. ACM. doi: 10.1145/3356464.3357702.

Ryan, M. (2010). Constraint-based multi-robot path planning. In *IEEE International Conference on Robotics and Automation, ICRA 2010, Anchorage, Alaska, USA, 3-7 May 2010*, pages 922–928. IEEE. doi: 10.1109/ROBOT.2010.5509582.

Sharon, G., Stern, R., Goldenberg, M. a Felner, A. (2013). The increasing cost tree search for optimal multi-agent pathfinding. *Artif. Intell.*, **195**, 470–495. doi: 10.1016/j.artint.2012.11.006.

Sharon, G., Stern, R., Felner, A. a Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artif. Intell.*, **219**, 40–66. doi: 10.1016/j.artint.2014.11.006.

Standley, T. (2010). Finding optimal solutions to cooperative pathfinding problems. In Fox, M. a Poole, D., editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press.

Stern, R., Sturtevant, N., Felner, A., Koenig, S., Ma, H., Walker, T. T., Li, J., Atzmon, D., Cohen, L., Kumar, T. K. S., Boyarski, E. a Barták, R. (2019). Multi-agent pathfinding: Definitions, variants, and benchmarks. In Surynek, P. a Yeoh, W., editors, *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019*, pages 151–159. AAAI Press.

Surynek, P. (2010). An optimization variant of multi-robot path planning is intractable. In Fox, M. a Poole, D., editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press.

Surynek, P. (2012). On propositional encodings of cooperative path-finding. In *IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, November 7-9, 2012*, pages 524–531. IEEE Computer Society. doi: 10.1109/ICTAI.2012.77.

Surynek, P. (2014). A simple approach to solving cooperative path-finding as propositional satisfiability works well. In Pham, D. N. a Park, S., editors, *PRICAI 2014: Trends in Artificial Intelligence - 13th Pacific Rim International Conference on Artificial Intelligence, Gold Coast, QLD, Australia, December 1-5, 2014. Proceedings*, volume 8862 of *Lecture Notes in Computer Science*, pages 827–833. Springer. doi: 10.1007/978-3-319-13560-1_66.

Surynek, P., Felner, A., Stern, R. a Boyarski, E. (2016). Efficient SAT approach to multi-agent path finding under the sum of costs objective. In Kaminka, G. A., Fox, M., Bouquet, P., Hüllermeier, E., Dignum, V., Dignum, F. a van Harmelen, F., editors, *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 810–818. IOS Press. doi: 10.3233/978-1-61499-672-9-810.

Wagner, G. a Choset, H. (2011). M*: A complete multirobot path planning algorithm with performance bounds. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2011, San Francisco, CA, USA, September 25-30, 2011*, pages 3260–3267. IEEE. doi: 10.1109/IROS.2011.6095022.

Wan, Q., Gu, C., Sun, S., Chen, M., Huang, H. a Jia, X. (2018). Lifelong multi-agent path finding in A dynamic environment. In *15th International Conference on Control, Automation, Robotics and Vision, ICARCV 2018, Singapore, November 18-21, 2018*, pages 875–882. IEEE. doi: 10.1109/ICARCV.2018.8581181.

Yu, J. a LaValle, S. M. (2013). Structure and intractability of optimal multi-robot path planning on graphs. In desJardins, M. a Littman, M. L., editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*. AAAI Press.

Yu, J. a LaValle, S. M. (2016). Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Trans. Robotics*, **32**(5), 1163–1177. doi: 10.1109/TRO.2016.2593448.

Zhou, N., Kjellerstrand, H. a Fruhman, J. (2015). *Constraint Solving and Planning with Picat*. Springer Briefs in Intelligent Systems. Springer. ISBN 978-3-319-25881-2. doi: 10.1007/978-3-319-25883-6.

Švancara, J., Vlk, M., Stern, R., Atzmon, D. a Barták, R. (2019). Online multi-agent pathfinding. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 7732–7739. AAAI Press. doi: 10.1609/aaai.v33i01.33017732.

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

ASP Answer set programming

BFS Breadth-first search (Prohledávání do šířky)

CBS Conflict-based search

CP Constraint programming (Programování s omezujícími podmínkami)

MAPD Multi-agent pickup and delivery

MAPF Multi-agent path finding

MIP Mixed-Integer programming (Smíšené lineární programování)

SAT Boolean satisfiability problem (Problém splnitelnosti booleovské formule)

A. Přílohy

A.1 Obsah elektronické přílohy

Elektronická příloha obsahuje:

- *online-MAPF* – adresář obsahující aplikaci online-MAPF
- *simulation* – adresář s implementací hierarchického algoritmu a experimentů
- *výsledky* – adresář s výsledky provedených experimentů

A.2 Uživatelská dokumentace k experimentům

Implementace experimentů byla vyvíjena a otestována na operačním systému Windows 10, na jiných operačních systémech testována nebyla.

A.2.1 Jak experimenty spustit

Pro spuštění experimentů je potřeba mít nainstalovaný programovací jazyk Python 3. Program používá kromě standardních knihoven také pythoní knihovny *joblib* a *matplotlib*. Před prvním spuštěním je tedy potřeba obě knihovny nainstalovat. Doporučený způsob instalace je pomocí pipu:

```
> pip install joblib matplotlib
```

Dále je potřeba nainstalovat programovací jazyk Picat – nejprve ze stránky http://picat-lang.org stáhnout archiv pro konkrétní operační systém, rozbalit stažený archiv a následně vložit spustitelný soubor s názvem *picat* do složky Picat. Pokud se spustitelný soubor nejmenuje přesně *picat*, je nutné jej přejmenovat.

Jakmile jsou obě pythoní knihovny a programovací jazyk Picat nainstalované, experimenty je možné (po přepnutí do složky Simulation) spustit pomocí následujícího příkazu:

```
> python3 main.py agents tasks frequency [nepovinné_parametry]
```

Příkaz očekává následující povinné parametry:

- *agents* – určuje počet agentů MAPD instance
- *tasks* – určuje počet úkolů, které mají agenti vykonat
- *frequency* – určuje frekvenci zveřejňování úkolů

Nepovinné parametry jsou následující:

- -a *algorithms* specifikuje kombinaci algoritmů použitých na globální a lokální úrovni pro druh experimentu simple. Může být *CBS*, *SAT*, *CBS+heat* nebo *SAT+heat*. Defaultní hodnota je nastavena na *CBS*.

- -t *experiment_type* specifikuje druh experimentu. Může být *division*, *algorithms* nebo *simple*. Defaultní hodnota je nastavena na *simple*.
- -i *instancies* specifikuje počet MAPD instancí, na kterých algoritmus poběží. Defaultní hodnota je 1.
- -l *large_map* určuje, zda se bude při experimentu používat větší mapa.
- -w *write_result* určuje, zda bude výsledek zapsán do souboru.
- -e *export_solution* určuje, zda se bude řešení jednotlivých MAPD instancí zapisovat do souborů (jeden soubor pro každou MAPD instanci). Pouze pro druh experimentu simple.

Například následující příkaz spustí hierarchický algoritmus používající klasický A* na globální úrovni a CBS na lokální úrovni na 3 různých MAPD instancích s 15 agenty, 250 úkoly a frekvencí zveřejňování úkolů 1:

```
> python3 main.py 15 250 1 -i 3
```

Vyexportované řešení MAPD instance (pomocí nepovinného parametru -e *export_solution*) je kompatibilní s aplikací online-MAPF (viz příloha A.3). Řešení je tedy možné si v aplikaci nechat zobrazit.

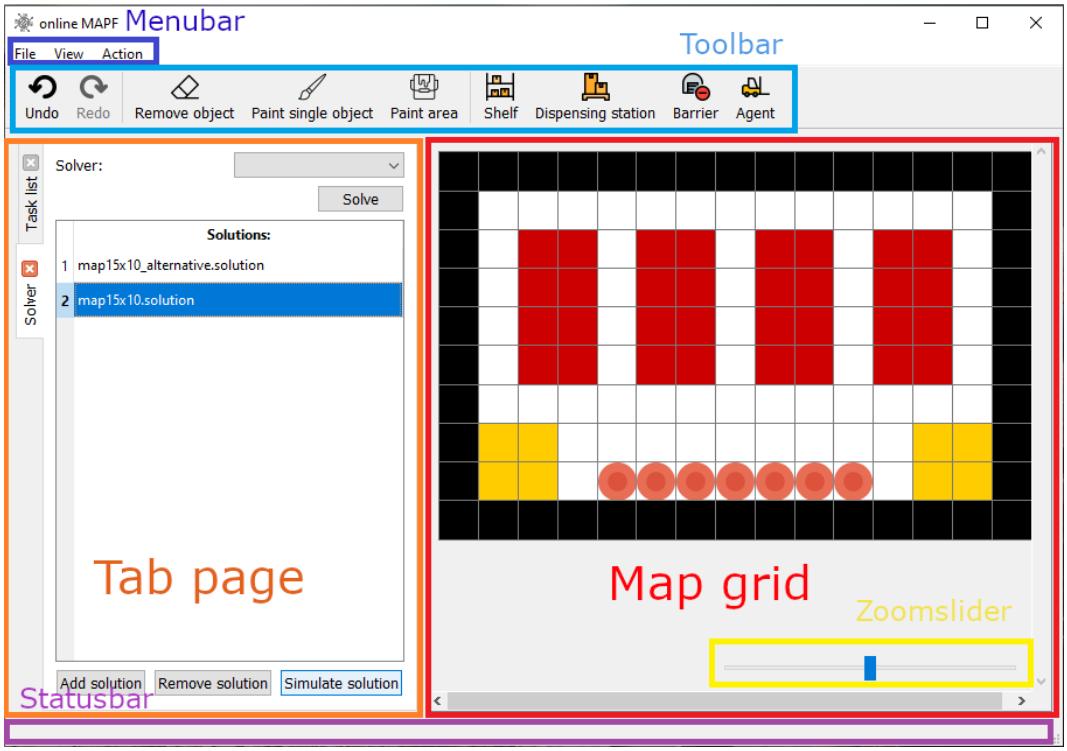
Program podporuje následující druhy experimentů:

- Experiment simple – spustí hierarchický algoritmus se zadanou kombinací algoritmů (specifikovaných parametrem -a *algorithms*) na zadaném počtu MAPD instancí.
- Experiment algorithm – experiment ze sekce 5.3. Spustí experiment porovnávající hierarchický algoritmus používající různou kombinaci algoritmů na globální a lokální úrovni na zadaném počtu MAPD instancí.
- Experiment division – experiment ze sekce 5.4. Spustí experiment porovnávající hierarchický algoritmus na různých rozděleních skladu na sektory na zadaném počtu MAPD instancí.

A.3 Uživatelská dokumentace online-MAPF

Online-MAPF je okenní aplikace napsaná v Pythonu, která slouží pro práci s MAPD problémy. Aplikace umožňuje definovat vlastní MAPD problémy a zobrazovat řešení MAPD problému v simulátoru řešení. Na obrázku A.1 jsou popsány jednotlivé části hlavního okna aplikace.

V horní části okna se nachází rozbalovací menu (na obrázku označeno jako Menubar) a panel nástojů (označeno jako Toolbar), ve spodní části se pak nachází stavový řádek (označeno jako Statusbar), který zobrazuje dodatečné infromace o ostatních ovládacích prvcích. Hlavní část okna je tvořena třemi základními prvky - lištou záložek (Tab page), mapou (Map grid) a posuvníkem přibližování (Zoomslider).



Obrázek A.1: Popis hlavního okna aplikace online-MAPF

A.3.1 Jak aplikaci spustit

Ke spuštění aplikace je potřeba mít nainstalovaný Python 3. Aplikace používá pythoní knihovnu *PyQt5*. Před prvním spuštěním je tedy potřeba tuto knihovnu nainstalovat. Doporučený způsob instalace je pomocí *pipu*:

```
> pip install pyqt5
```

Jakmile je knihovna PyQt5 úspěšně nainstalovaná, můžete aplikaci spustit zadáním následujícího příkazu:

```
> python3 main.py
```

A.3.2 Definování MAPD problému

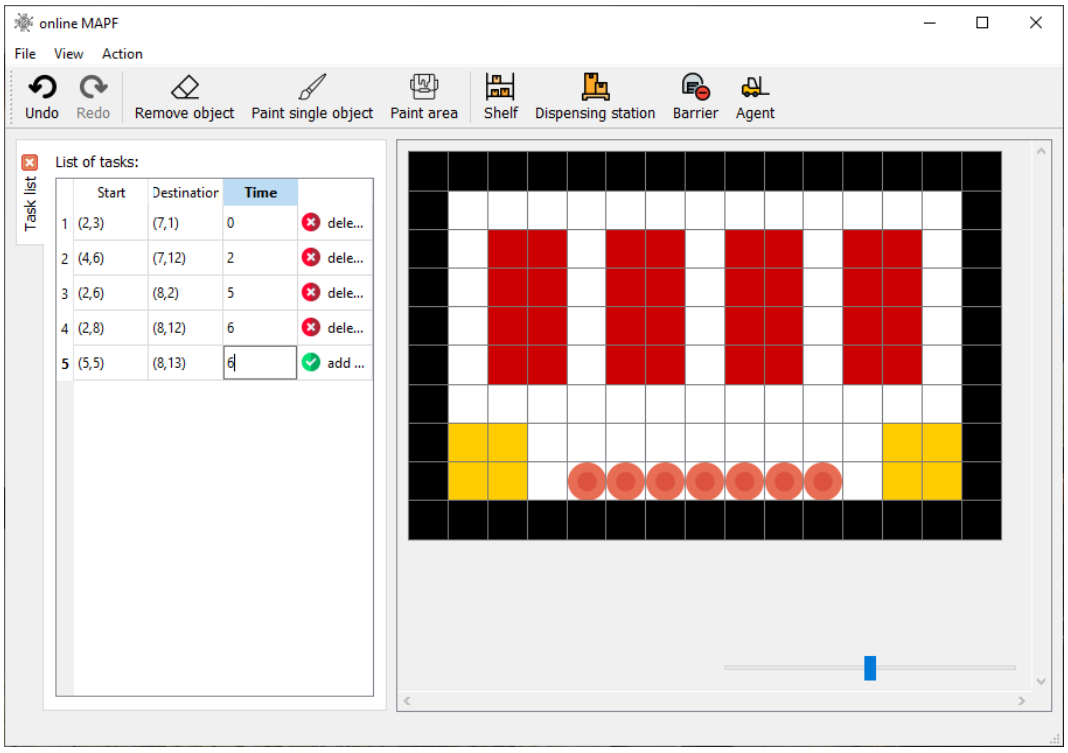
Tvorba mapy

Chcete-li vytvořit novou mapu, vyberte v nabídce Soubor možnost *New map* nebo stiskněte klávesovou zkratku *Ctrl+N*. Po zadání požadovaných rozměrů mapy se vytvoří prázdná mapa. Mapu je možné přiblížit či oddálit kolečkem myši nebo pomocí posuvníku, který se nachází v pravé dolní části okna.

Tvorba mapy funguje ve stylu klasického malování - nejprve je potřeba vybrat nástroj a objekt z Panelu nástrojů. Každý objekt má svou vlastní specifickou barvu. S nástroji *Remove objekt* a *Paint single object* následně stačí kliknout na políčko mapy, na které chcete vybraný nástroj aplikovat. V případě nástroje *Paint area* klikněte a táhněte pro vybarvení oblasti. Na levé straně Panelu nástrojů se nacházejí tlačítka *Undo* (vrátit změnu) a *Redo* (znovu udělat změnu), které

umožňují vrátit nebo opakovat provedené akce v pořadí, v jakém byly akce provedeny.

Mapu můžete uložit pro pozdější použití pomocí možnosti *Save map* v nabídce File nebo pomocí klávesové zkratky *Ctrl+S*. Uložená mapa neobsahuje informace o agentech, ty jsou uloženy v samostatném souboru. Pro uložení agentů vyberte možnost *Save agents* v nabídce File. Načíst existující mapu lze vybráním možnosti *Load map* v nabídce File nebo pomocí klávesové zkratky *Ctrl+L*. Pro načtení agentů pak vyberte možnost *Load agents* v nabídce File.



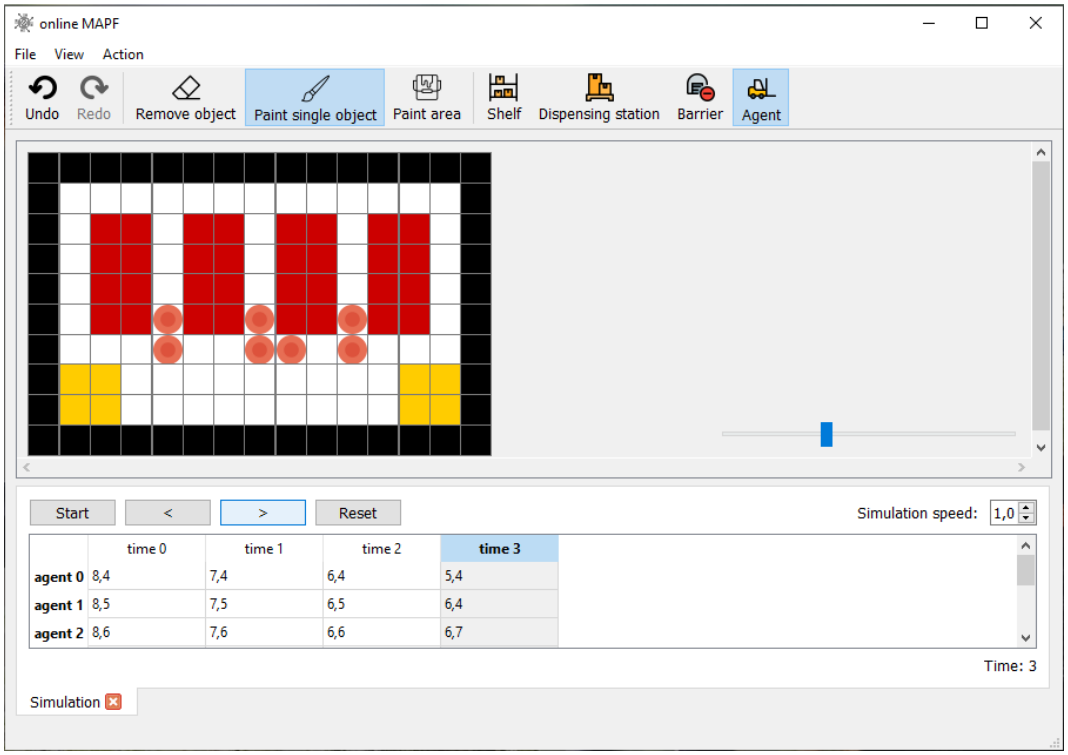
Obrázek A.2: Okno aplikace online-MAPF s otevřeným editorem úkolů

Zadávání úkolů

Když je mapa hotová, můžete začít zadávat úkoly pro agenty na dané mapě. Seznam připravených úkolů můžete nalézt v liště záložek v záložce úkoly. Pro zobrazení záložky úkolů vyberte možnost *Task list* v nabídce View nebo stiskněte klávesovou zkratku *Ctrl+T*. Okno aplikace s otevřeným editorem úkolů je zobrazeno na obrázku A.2.

Pro přidání nového úkolu do seznamu je nejprve potřeba vyplnit všechny sloupečky posledního řádku v seznamu úkolů a následně stisknout tlačítko *add task*. Když vybíráte startovní a cílové souřadnice, můžete je buď vyplnit ručně, nebo můžete vybrat souřadnici kliknutím na konkrétní políčko mapy. Dříve zadané úkoly není možné editovat, mohou být pouze odebrány pomocí tlačítka *delete task*.

Seznam úkolů můžete uložit pro pozdější použití pomocí možnosti *Save tasks* v nabídce File. Pro načtení dříve uložených úkolů použijte možnost *Load tasks* opět ve File nabídce.



Obrázek A.3: Okno simulace řešení v aplikaci online-MAPF.

A.3.3 Zobrazení řešní MAPD problému

Záložka řešení umožňuje přidávat řešení definovaného MAPD problému. Pro otevření záložky řešení vyberte *Solve* možnost v nabídce Action nebo zmáčkněte klávesovou zkratku *Ctrl+O*. Kliknutím na tlačítko *Add solution* můžete nahrát existující řešení MAPD problému. Přidané řešení se následně objeví v seznamu řešení. Řešení je možné odebrat tlačítkem *Remove solution* nebo přehrát kliknutím na tlačítko *Simulate solution*.

Simulace řešení

Po stisknutí tlačítka *Simulate solution* se automaticky otevře záložka simulace. Záložku jde otevřít i manuálně vybráním možnosti *Simulate* v nabídce Action nebo klávesovou zkratkou *Ctrl+I*. Okno aplikace s otevřenou záložkou simulace je zobrazeno na obrázku A.3.

Hlavní část záložky simulace je tvořena tabulkou agentů, která ukazuje pozice všech agentů v čase. Tlačítka nad tabulkou slouží k ovládání simulace. Tlačítko *Play* začne přehrávat simulaci řešení. Tlačítko *Reset* vrací simulaci na začátek (čas 0). Tlačítka $>$ a $<$ slouží k posouvání simulace o jeden krok vpřed, respektive vzad.