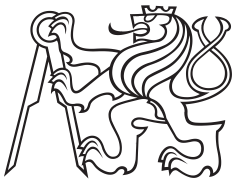
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

Diagnostika realizace plánů pro multiagentní plánování

Denisa Mužíková

Vedoucí: Ing. David Zahrádka
Studijní program: Otevřená informatika
Specializace: Základy umělé inteligence a počítačových věd
Květen 2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Mužíková** Jméno: **Denisa** Osobní číslo: **507314**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Otevřená informatika**

Specializace: **Základy umělé inteligence a počítačových věd**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Diagnostika realizace plánů pro multiagentní plánování

Název bakalářské práce anglicky:

Execution Diagnostics for Multi-Agent Pathfinding

Pokyny pro vypracování:

1. Seznamte se s metodami Safe Interval Path Planning (SIPP) pro plánování v dynamickém prostředí a Action Dependency Graph (ADG) pro robustní realizaci plánů.
2. Implementujte metodu ADG.
3. Navrhněte a implementujte metodu, která sleduje vývoj bezpečných intervalů (safe intervals) během realizace plánů, detekuje možné kolize a klasifikuje je dle jejich závažnosti.
4. Proveďte experimentální ověření vlastností navržené metody a pozorované vlastnosti okomentujte.
5. Navrženou metodu popište a kód zdokumentujte.

Seznam doporučené literatury:

[1] Phillips, Mike, and Maxim Likhachev. "Sipp: Safe interval path planning for dynamic environments." 2011 IEEE international conference on robotics and automation. IEEE, 2011.
[2] Hönig, Wolfgang, et al. "Persistent and robust execution of MAPF schedules in warehouses." IEEE Robotics and Automation Letters 4.2 (2019): 1125-1131.
[3] Ma, Hang, and Sven Koenig. "AI buzzwords explained: Multi-agent path finding (MAPF)." AI Matters 3.3 (2017): 15-19.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. David Zahrádka inteligentní a mobilní robotika CIIRC

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

Datum zadání bakalářské práce: **03.02.2024** Termín odevzdání bakalářské práce: **24.05.2024**

Platnost zadání bakalářské práce: **21.09.2025**

Ing. David Zahrádka	prof. Dr. Ing. Jan Kybic	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

. Datum převzetí zadání

Podpis studentky

Poděkování

Především bych chtěla poděkovat svému vedoucímu Ing. Davidu Zahrádkovi za ochotu a pomoc v průběhu vedení této práce.

Ráda bych také poděkovala své rodině a přátelům za podporu během celého studia.

Prohlášení

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

V Praze, 24. května 2024
Denisa Mužíková

Abstrakt

Tato bakalářská práce se zabývá problematikou multi-agentního plánování a zaměřuje se na robustní exekuci plánů reálnými roboty. Cílem práce je minimalizovat dobu exekuce plánů. Pro zajištění robustní exekuce je využita metoda Action Dependency Graph, která je v této práci rozšířena o metody pro sledování průběhu exekuce a predikci kolizí. Optimalizace doby exekuce je realizována vhodným přeplánováním na základě těchto predikcí. Účinnost navržených metod je v práci ověřena v simulacích.

Klíčová slova: multi-agentní plánování, robustní exekuce, graf akčních závislostí, přeplánování

Vedoucí: Ing. David Zahrádka

Abstract

This bachelor thesis focuses on Multi-Agent Pathfinding and on robust execution of plans by real robots. The goal of this thesis is to minimize execution time. To ensure robust execution, the Action Dependency Graph (ADG) is used. In this thesis ADG is extended by methods for tracking execution progress and predicting collisions. The optimization of execution time is realized by replanning based on these predictions. The effectiveness of proposed methods is verified in simulations.

Keywords: Mutli-Agent Pathfinding, robust execution, Action Dependency Graph, replanning

Title translation: Execution Diagnostics for Multi-Agent Pathfinding

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Stále se zvyšuje využití robotů nejen v průmyslu, ale také ve službách. V průmyslu jsou roboti využíváni například na výrobních linkách, ve službách pak v autonomních skladech. V prvním případě může být výrobní linka pevně (či částečně dynamicky) nastavena pro přesně dané operace, v druhém případě je při využití skupiny robotů potřeba robotům přidělit úkoly a naplánovat jejich trasy.

Úloha plánování pohybu skupiny robotů může být formulována jako problém multi-agentního plánování (MAPF; z angl. Multi-Agent Pathfinding)[11]. Tento problém se zabývá nalezením bezkolizních cest pro skupiny robotů. Cílem je nalezení takového plánu, který minimalizuje dané kritérium (např. celkový čas) a zároveň neobsahuje kolize. Ukázka plánu je na obrázku 1.1.

Aktuálně používané přístupy k řešení problému MAPF jsou schopné rychle nalézt bezkolizní plány. Existují metody, které dokáží najít optimální plán. Jiné metody dokáží najít suboptimální plán, některé s garantovanou mezí optimality. Podle klasické definice problému MAPF jsou akce robotů vykonávány synchronně a v diskrétním čase, kde jedna akce trvá přesně jeden časový úsek. Synchronizace robotů ale není vždy možná. To vede k problémům například v případě following konfliktu, kde by podle plánu měl jeden robot opouštět svou pozici a druhý na ni přijíždět. Pokud by akce prvního robota začala být prováděna později, než akce robota následujícího, došlo by ke kolizi. [11]

Při exekuci těchto plánů na reálných robotech můžeme narazit i na další problémy, které klasický problém MAPF nezohledňuje. Příkladem může být zpoždění robotů (ať už z důvodu neideálního pohonu) nebo zjištění neočekávané překážky v prostoru, kde se roboti pohybují. Nejen, že je od toho momentu plán neplatný, ale navíc může dojít ke kolizi robotů.

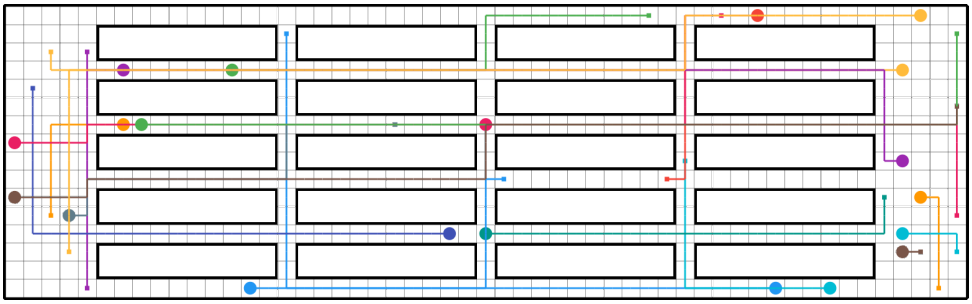
Problémy vzniklé zpožděním robotů lze vyřešit pomocí metod pro robustní exekuci. Příkladem může být metoda Action Dependency Graph (ADG)[4]. Tato metoda zachovává pořadí průjezdu robotů křižovatkami dle plánu MAPF, a tím zajišťuje jeho bezkolizní exekuci. Nevýhodou metody je, že v některých případech musí pro zajištění robustní exekuce zpozdit některé roboty.

Algoritmus Safe Interval Path Planning (SIPP)[9] představuje plánování s bezpečnými intervaly. Robot má pro každou pozici v prostoru bezpečné intervaly, kdy se může na dané pozici vyskytovat. Použití tohoto přístupu umožňuje plánovat s dynamickými překážkami, pokud známe nebo jsme

schopni odhadnout jejich trajektorie. Při sledování vývoje bezpečných intervalů v čase můžeme získat přehled o prostoru a průběhu exekuce a detekovat tak blížící se kolize.

Tato bakalářská práce se zaměřuje na provádění plánů v multi-agentních systémech reálných robotů s cílem minimalizovat celkovou dobu exekuce plánů. Toho se snažíme dosáhnout pomocí predikce kolizí při vykonávání plánu a jejich následným řešením za pomocí přeplánování.

Práce je strukturována následujícím způsobem: kapitola 2 obsahuje rešerši odborné literatury týkající se problému robustní exekuce MAPF. Kapitola 3 definuje problém MAPF a pojmy a problémy s ním spojené a řešené v této práci. V kapitole 4 je popsána metoda ADG pro bezkolizní provádění plánu. Tato metoda je v kapitole rozšířena o predikci časů exekuce konkrétních akcí robotů a predikci kolizí. Práce pokračuje kapitolou 5, kde je popsán simulátor exekuce plánů MAPF, který je pak použit k provedení experimentů v kapitole 7. V kapitole 6 je popsána implementace metod do simulátoru a v kapitole 7 jsou výsledky experimentálního testování implementovaných metod pro řešení kolizí pomocí přeplánování. V poslední kapitole 8 je shrnutí výsledků práce.



Obrázek 1.1: Ukázka plánu agentů pro pohyb v autonomním skladu.

Kapitola 2
Rešerše literatury

Multi-Agent Pathfinding (MAPF) je problém hledání cest pro více agentů, kde hlavním cílem je vytvořit plán bez kolizí mezi jednotlivými agenty. [7] [11] Problém MAPF lze řešit například pomocí metody Safe Interval Path Planning (SIPP)[9], která pomocí bezpečných intervalů (safe interval) umožňuje plánování pro jednoho agenta s dynamickými překážkami. Bezpečný interval je časový úsek po který se může agent vyskytovat na konkrétní pozici, aniž by došlo ke kolizi. Metoda SIPP s využitím algoritmu A^*, hledá pro agenta cestu tak, aby byla naplánována pouze do bezpečných intervalů a předešlo se tak kolizím s dynamickými překážkami. Ačkoliv je SIPP metodou pro hledání cesty jednoho agenta, používá se pro řešení problému MAPF v podobě prioritizovaného plánovače. Nejprve naplánuje cestu pro agenta s nejvyšší prioritou. Poté hledá cestu pro agenta s nižší prioritou, kde agenty s vyšší prioritou považuje za dynamické překážky, kterým je třeba se vyhnout. Takto jsou postupně naplánovány cesty pro všechny agenty.

Jiný přístup pro tvorbu plánů nabízí Conflict Based Search (CBS)[10]. Jde o dvouúrovňový algoritmus, který pomocí řešení konfliktů mezi plány agentů najde optimální plán bez konfliktů. Ve vyšší úrovni prohledává Constraint Tree (CT), což je binární strom ve kterém každý vrchol obsahuje množinu omezení, kde se který agent nesmí nacházet ve který časový úsek, plán pro každého agenta, který respektuje daná omezení a součet cen plánů jednotlivých agentů. Vždy když je ve vrcholu nalezen konflikt, tak je strom rozvětven a v každé větvi je přidáno jedno jiné omezení, které zamezí nalezenému konfliktu. V nižší úrovni jsou hledány cesty pro jednotlivé agenty tak, aby byla splněna daná omezení. Vrchol CT je validní, pokud nejsou v plánu agentů konflikty. Optimalita řešení z hlediska *sum of costs* je zajištěna díky prohledávání CT pomocí best-first search podle ceny vrcholů.

Vylepšením pro algoritmus CBS je suboptimální Enhanced Conflict Based Search (ECBS)[1], který má garantovanou hranici suboptimality w, která je parametrem algoritmu. Je pak zaručeno, že nalezené řešení nemá více než w-krát větší cenu než optimální řešení. Díky rozvolnění podmínky pro optimalitu je metoda ECBS schopna nalézt řešení výrazně rychleji a také s vyšší hranicí suboptimality vyřešit problémy, které CBS nevyřeší.

Bezkolizním prováděním plánu se zabývají metody robustní exekuce. Příkladem je metoda Action Dependency Graph (ADG)[4], která při exekuci

zachovává pořadí akcí robotů a zároveň zajišťuje, aby pohyb robota na pozici, kde se nacházel jiný robot nastala až po dokončení odjezdu z dané pozice. ADG tak umožňuje exekuci celého plánu bez potřeby přeplánování.

Exekuci na reálných robotech lze také provést pomocí metody MAPF-POST[3], která z MAPF plánu pomocí Simple Temporal Network (STN) vytvoří plán pro exekuci. Tato metoda navíc počítá s kinematickými omezeními robotů. Na vstupu metody je MAPF plán ze kterého se vytvoří Temporal Plan Graph (TPG). To je acyklický orientovaný graf $G = (V, E)$, kde každý vrchol $v_k^i \in V$ reprezentuje vjezd agenta i na k-tou pozici v jeho plánu. Každá hrana zobrazuje závislost mezi dvěma vrcholy. Závislosti jsou dvou typů. Hrana $e = (v_k^i, v_{k+1}^i)$ je typu 1 a uchovává pořadí akcí dle plánu jednotlivých agentů. Hrany typu 2 jsou hrany $e = (v_k^i, v_l^j)$, kde vjezd na pozici reprezentovanou vrcholem v_l^j je možný, až potom, co bude dokončena akce spojená s v_k^i. Následně je graf rozšířen o další vrcholy zvané safety markers, aby byla zajištěna bezpečná vzdálenost mezi agenty. Z rozšířeného TPG je poté vytvořena STN. To je orientovaný acyklický graf $G' = (V', E')$, kde každá hrana $e = (v, v') \in E'$ má hranice $LB(e)$ a $UB(e)$, které značí, že akce v musí být naplánována n časových jednotek před akcí v', kde $n \in [LB(e), UB(e)]$. Tyto hranice nám umožňují vyjádřit různé doby trvání jednotlivých akcí například v případech, kdy různí agenti mají odlišnou maximální rychlost nebo pokud hrany reprezentují různě dlouhé cesty. Z STN je následně například pomocí lineárního programování vytvořen plán exekuce, který udává, kdy má být dokončena každá akce. MAPF-POST snižuje množství potřebných přeplánování při provádění plánu, jelikož v případě porušení plánu stačí pouze vytvořit novou STN pro zbývající akce v plánu.

Výhoda metody ADG je, že je jednoduchá a pro realizaci plánu stačí na začátku vytvořit ADG z plánu. Oproti tomu metoda MAPF-POST umožňuje do plánu exekuce zahrnout kinematická omezení robotů a vytváří plán exekuce, který může například počítat s tím, že někteří roboti jsou pomalejší než jiní nebo zahrnout doby otáčení do plánu exekuce. Nevýhodou je, že pro vytvoření STN je třeba vyřešit minimalizační úlohu, což může být pro složitější plány výpočetně náročný proces.

Metoda ADG pro exekuci plánu zachovává pořadí průjezdu křižovatek dle původního MAPF plánu. To může vést ke zpoždění více robotů například v případě, že se nějaký robot zpozdí. Prohození pořadí některých agentů však může vést k deadlocku. Pro možnost prohození některých závislostí agentů je možné vytvořit Switchable Action Dependency Graph (SADG)[2], který zachycuje možné prohození pořadí agentů. SADG rozšiřuje ADG o možnost přeřazení pořadí agentů a stále garantuje bezkolizní exekuci MAPF plánu.

Další možnost pro bezkolizní exekuci a možnost změnit pořadí agentů je Bidirectional Temporal Plan Graph (BTPG)[12]. BTPG rozšiřuje TPG o obousměrné páry hran (bidirectional pairs) (e, e'). Jedná se o hrany typu 2 $e = (v_{k+1}^i, v_l^j)$ a $e' = (v_{l+1}^j, v_k^i)$. Při exekuci je robotovi j povolen přesun na další pozici v plánu, pokud pro všechny hrany $e = (v_k^i, v_l^j)$ typu 2 platí, že robot i již navštívil pozici reprezentovanou vrcholem v_k^i. Výjimku tvoří hrany z obousměrného páru, kde je vždy platná jen jedna hrana, a to ta, která

umožní projet prvnímu robotovi, který chce projet křižovatkou.

Jinou možností pro robustní exekuci může být oprava plánu v případě, že se plán stane neplatným z důvodu zpoždění robota. Pro opravu plánu je možné použít plánovač MAPF a vytvořit nový plán ze současných pozic robotů. V závislosti na zvoleném plánovači může být hledání nového plánu výpočetně náročným procesem. K jeho urychlení lze pro vytvoření nového plánu jako mapu použít Constrained Graph (CG) nebo Improved Constrained Graph (ICG)[6]. Tyto grafy jsou vytvořeny z původního plánu MAPF a umožňují robotovi pohyb pouze po cestě z původního plánu. S využitím CG nebo ICG je tímto způsobem vytvořen nový plán, kde cesty robotů jsou stejné, jen je přidáno zpoždění, aby byl plán platný.

Kapitola 3
Formulace problému

V této kapitole jsou definovány problémy, kterými se zabývá tato práce. Sekce 3.1 definuje problém MAPF[11] a s ním související pojmy a sekce 3.2 definuje exekuci plánu.

3.1 Formulace MAPF

MAPF problém s A agenty je trojice $\langle G, s, d\rangle$, kde $G = (V, E)$ je neorientovaný graf, $s : [1, \ldots, A] \to V$ přiřazuje každému agentovi počáteční vrchol a $d : [1, \ldots, A] \to V$ každému agentovi přiřazuje cíl. Cílem problému MAPF je nalézt bezkolizní cesty pro všechny agenty tak, aby dorazili ze startu do svých cílů.

Formulace MAPF problému počítá s diskrétním časem. V jeden čas se agent nachází v jednom vrcholu grafu a může provést jednu akci $a : V \to V$, kde akce trvá jeden časový krok. Rozlišujeme dva typy akcí $a(v) = v'$: čekání, při které $v = v'$ a pohyb, kde $v \neq v'$. Pro každou akci pohybu platí, že $(v, v') \in E$. Agenti se pohybují synchonizovaně dle časových kroků. Agenti tedy všichni zároveň splní první akci z plánu, pak druhou a tak pokračují, než budou vykonány všechny akce z plánu.

Plán agenta $\pi_i = (v_0, \ldots, v_n)$ je posloupnost vrcholů, která udává pozici agenta $\pi_i[t] \in V$ pro každý diskrétní čas $t = 0, \ldots, n$. Pro každý plán agenta i platí, že $v_0 = s(i)$, $v_n = d(i)$ a pro každé dva vrcholy v_k a v_{k+1} existuje akce $a(v_k) = v_{k+1}$. Délka plánu T_{π_i} je počet časových kroků potřebných k vykonání plánu agenta i a je rovna počtu akcí, které agent musí vykonat, než splní svůj plán. Řešení problému MAPF je plán $P = \{\pi_i \mid i = 1, \ldots, A\}$.

3.1.1 Konflikty mezi plány

Při řešení problému MAPF mohou v plánu nastat konflikty mezi agenty. Plán MAPF se nazývá platným, pokud mezi žádnými plány agentů nenastane konflikt. Pro potřeby práce uvažujeme následující typy konfliktů:

- *Vertex konflikt* mezi dvěma plány agentů π_i a π_j nastane tehdy, pokud by podle plánu oba agenti měli být v jednom čase ve stejném vrcholu. Tedy: $\exists t : \pi_i[t] = \pi_j[t]$.

- *Swap konflikt* mezi dvěma plány π_i a π_j nastane tehdy, pokud by si agenti během změny jednoho časového kroku měli vyměnit pozice. Tedy: $\exists t : \pi_i[t] = \pi_j[t+1], \pi_j[t] = \pi_i[t+1]$.
- *Cycle konflikt* mezi plány agentů $\pi_i, \pi_{i+1}, \ldots, \pi_j$ nastává tehdy, pokud platí $\pi_i[t+1] = \pi_{i+1}[t]$, $\pi_{i+1}[t+1] = \pi_{i+2}[t]$, $\ldots$, $\pi_j[t+1] = \pi_i[t]$ pro nějaký čas t.
- *Following konflikt* nastane tehdy, pokud má agent vjet na pozici ve které se v předchozím časovém kroku nacházel jiný agent. Tedy pokud pro dva agenty i a j $\exists t : \pi_i[t] = \pi_j[t+1]$.

3.1.2 Chování agentů v cíli

V plánu MAPF mohou agenti do svých cílů dorazit v různý čas, proto je třeba definovat co se s agentem stane, když do cíle dorazí. První z možností je, že agent v moment kdy dorazí do cíle zmizí. Druhá varianta předpokládá, že agent zůstává v místě cíle a blokuje jej. V této práci uvažujeme variantu, kde agenti v cíli zůstávají.

3.1.3 Účelové funkce

Pro jedno zadání problému MAPF může existovat více různých plánů, které daný problém řeší. Účelové funkce slouží k hodnocení a porovnávání těchto plánů mezi sebou. Využívají se pak v optimalizačních úlohách, kde je cílem najít plán s co nejmenší hodnotou účelové funkce. Nejčastěji využívané účelové funkce v problému MAPF jsou *makespan* a *sum of costs*:

- *Makespan* představuje počet časových kroků potřebných k vykonání celého plánu. *Makespan* plánu P je definován jako $\max_i\{T_{\pi_i}\}$.
- *Sum of costs* je součet časových kroků potřebných k vykonání plánu každého agenta. *Sum of costs* plánu P se rovná $\sum_i T_{\pi_i}$.

3.1.4 Mapa

Mapa je tvořena 2D mřížkou. Každá buňka mřížky reprezentuje buď jeden vrchol grafu G nebo překážku. Akce pohybu z jedné buňky jsou možné do 4 sousedících buněk, kromě případů kde je sousedem překážka nebo pokud se buňka nachází na okraji mapy.

3.1.5 Rotace agentů

Některé varianty MAPF uvažují kromě pozice agenta i jeho natočení. V těchto případech je mapa 2D mřížka ze sekce 3.1.4. Natočení agenta je směr $o \in \{\text{sever}, \text{jih}, \text{západ}, \text{východ}\}$. Akce agenta jsou čekání, pohyb rovně a otočení o 90 stupňů v libovolném směru. [5] V práci uvažujeme variantu MAPF bez akcí rotace.

3.2 Exekuce plánu

Exekucí plánu rozumíme vykonání plánu MAPF skupinou simulovaných nebo reálných robotů. Každý robot má frontu akcí, které postupně vykonává. Pokud má robot prázdnou frontu, tak stojí. Jakmile dokončí akci hlásí její splnění. Akce jsou do fronty robota postupně přidávány. Předpokládáme, že robot vykoná každou akci zadanou ve frontě a to v čase, který se může lišit od předpokládaného času, přičemž délky akcí se mohou také lišit.

Exekuci plánu považujeme za robustní, pokud nedochází ke kolizím mezi jednotlivými roboty ani v případě, kdy se doba trvání akcí liší od předpokládané. [4]

3.2.1 Doba exekuce

Pro každého robota i definujeme dobu exekuce τ_i plánu π_i jako dobu od startu exekuce do nahlášení splnění poslední akce z π_i. Pro celou exekuci definujeme celkový čas exekuce $T_{max} = \max_i\{\tau_i\}$ a součet časů exekuce $T_{sum} = \sum_i \tau_i$. Pokud by exekuce probíhala přesně podle MAPF plánu, tak by hodnotě T_{max} odpovídal *makespan* a hodnotě T_{sum} by odpovídala *sum of costs*.

Problém řešený v této práci je inspirován autonomními sklady, kde úkoly robotům bývají přiřazovány postupně a očekává se, že po splnění úkolu může být robotovi zadána další práce. Je tedy žádoucí, aby každý robot dorazil do svého cíle co nejdříve, jde tedy o minimalizaci času T_{sum}.

Kapitola 4
Metody

Tato kapitola se zabývá metodami pro robustní exekuci a úpravou plánu s cílem minimalizovat dobu exekuce. Sekce 4.1 popisuje metodu ADG[4] pro zajištění robustní exekuce plánu. V sekci 4.2 rozdělujeme typy kolizí mezi roboty při robustní exekuci pomocí ADG. V sekci 4.3 metodu ADG rozšiřujeme o výpočet předpokládaných časů exekuce jednotlivých akcí. Sekce 4.4 definuje *metriky exekuce*, které slouží k sledování stavu exekuce a detekci potřeby přeplánování.

4.1 Action Dependency Graph

Metoda ADG zajišťuje robustní exekuci plánů. ADG je acyklický orientovaný graf $G = (V, E)$, který vyobrazuje návaznosti akcí v plánu. Vrchol $a_k^i \in V$ reprezentuje k-tou akci robota i. Každá akce je dvojice (p_1, p_2), kde p_1 je počáteční a p_2 koncová pozice dané akce. Závislosti mezi akcemi jsou dané hranami grafu. Pro každou hranu $(a, a') \in E$ platí, že akce a' nesmí být vykonána před a. Hrany $e \in E$ jsou dvojího typu, první typ jsou hrany $e = (a_k^i, a_{k+1}^i)$ mezi akcemi jednoho robota a ukládají tedy informaci o plánovaném pořadí akcí daného robota. Druhý typ jsou hrany $e = (a_k^i, a_l^j)$ mezi akcemi různých robotů a reprezentují jejich závislost.

4.1.1 Vytváření ADG

ADG je vytvářeno z MAPF plánu P a jeho tvorba probíhá ve dvou fázích podle algoritmu 1.

V první části algoritmu jsou vytvořeny všechny vrcholy a hrany typu 1. Pro každého agenta i (řádek 1) nejprve procházíme všechny jeho časové kroky t (řádek 2) a pro každé dva po sobě jdoucí vrcholy $\pi_i[t-1]$, $\pi_i[t]$ z plánu π_i je vytvořena akce a_t^i se startem ve vrcholu $\pi_i[t-1]$ a koncem $\pi_i[t]$ (řádek 3). Následně je akce přidána jako nový vrchol do ADG (řádek 4). Pokud a_t^i není první akce robota, tak je do grafu přidána hrana spojující předchozí akci robota s touto akcí (řádek 6).

V druhé fázi jsou do grafu přidány hrany typu 2, které reprezentují závislosti mezi akcemi. Pro jejich nalezení popořadě procházíme všechny možné dvojice akcí různých robotů (řádky 7-12). Hrana z vrcholu a_n^i do vrcholu a_m^j je do

Algoritmus 1 Tvorba ADG

Input: Plán P
Output: ADG $G = (V, E)$

```
   {Vytvoření všech vrcholů a hran typu 1}
 1: for i ← 1 to A do
 2:   for t ← 1 to T_{π_i} do
 3:     a_t^i ← (π_i[t − 1], π_i[t])
 4:     Add a_t^i to V
 5:     if t > 1 then
 6:       Add (a_{t−1}^i, a_t^i) to E
   {Vytvoření hran typu 2}
 7: for i ← 1 to A do
 8:   for n ← 1 to T_{π_i} do
 9:     for j ← 1 to A do
10:       if i = j then
11:         continue
12:       for m ← 1 to T_{π_j} do
13:         if start a_n^i = konec a_m^j and n ≤ m then
14:           Add (a_n^i, a_m^j) to E
15:           break
```

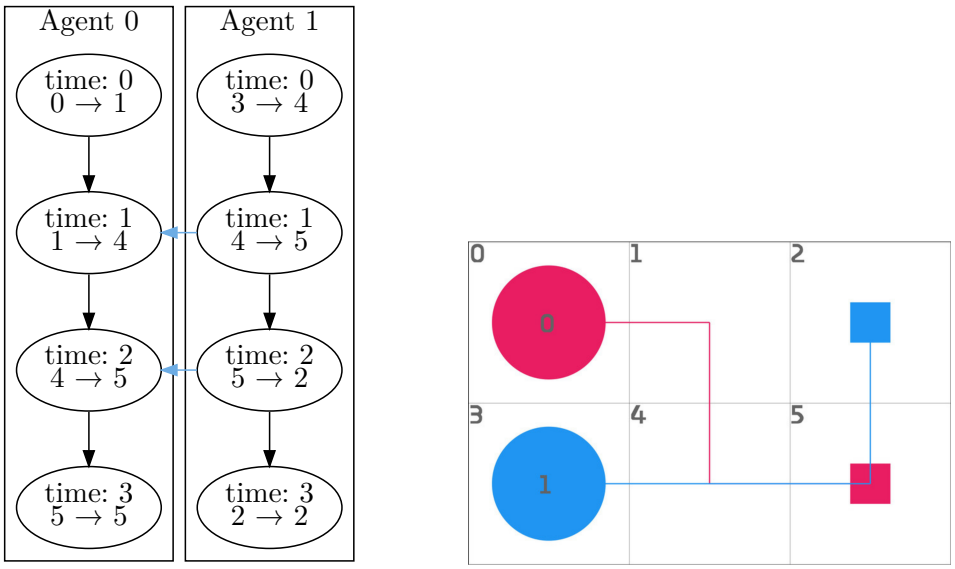
grafu přidána, pokud akce a^i_n začíná na stejné pozici, kde končí a^j_m a zároveň akce a^i_n neměla dle plánu P být vykonána později než a^j_m (řádky 13 a 14). Pokud již z vrcholu a^i_n vede hrana do některého vrcholu robota j, tak další hrana z vrcholu a^i_n do jiného vrcholu robota j není přidána, jelikož závislost kterou by reprezentovala je již zajištěna díky existující hraně typu 2 a hranám typu 1 (řádek 15).

4.1.2 Exekuce plánu s využitím ADG

K realizaci plánu ukládáme pro každý vrchol $a^i_k \in V$ jeho stav. Do fronty akcí robota i je přidána akce a^i_k, pokud akce a^i_{k-1} je již ve frontě a zároveň pro všechny hrany (a^j_l, a^i_k), kde $i \neq j$ platí, že akce a^j_l již byla dokončena.

Pro bezpečnou exekuci s využitím ADG je třeba, aby graf byl acyklický. Pokud je v grafu cyklus, znamená to *cycle konflikt* mezi plány jednotlivých agentů a znemožňuje zaručení bezkolizní exekuce. Naopak *following konflikt* v plánu, ze kterého je vytvořeno ADG, není problém, jelikož pravidla pro exekuci ADG zajistí, aby druhý robot mohl vjet na pozici až poté, co první robot pozici opustí.

Na obrázku 4.1a je ukázka ADG pro dva roboty s plány z obrázku 4.1b. Svislé hrany jsou hrany prvního typu a vodorovné hrany jsou druhého typu. Z grafu můžeme vidět, že po spuštění exekuce mohou oba roboti provést první akci z plánu. Roboti jsou nyní na pozicích 1 a 4. První hrana druhého typu od shora, že než agent 0 bude moct vykonat akci $1 \rightarrow 4$, tak musí agent 1 vykonat akci $4 \rightarrow 5$. Druhá hrana zachycuje situaci, kde roboti jedou za sebou a agent 0 bude moct pokračovat až poté, co agent 1 opustí pozici 5.



(a) : ADG **(b) :** Plán agentů

Obrázek 4.1: Ukázka ADG pro plán dvou agentů.

4.2 Klasifikace kolizí

S využitím ADG pro exekuci plánu je garantováno, že nedojde ke kolizím mezi jednotlivými roboty vykonávajícími plán. Zpoždění jednoho robota však může vést ke zpoždění dalších robotů. To jsou situace, kterým chceme předcházet. V následujícím textu pod pojmem kolize uvažujeme situace, kdy je robot blokován z důvodu závislosti na jiném robotovi.

Kolize dělíme dle vzdálenosti kolize od aktuálního stavu exekuce do dvou kategorií na kolize aktuální a budoucí:

- Aktuální kolize představují situace, kdy je jeden robot blokován druhým. Tyto situace jsou nejnaléhavější a vyžadují okamžité řešení, aby se minimalizovala ztráta času.

- Kolize v budoucnosti jsou naopak prediktivní a je možné je detekovat předem na základě plánovaných tras robotů. Tyto situace naznačují potenciální konflikty v pohybu robotů. Detekce těchto kolizí umožňuje předcházení budoucím konfliktům a optimalizaci plánů pohybu robotů.

Zabránit případným kolizím a zpožděním lze pomocí přeplánování. Nicméně existují i kolize, které přeplánováním řešit nelze. Situace kdy přeplánovat nelze mohou být způsobeny nedostatkem prostoru, přítomností pevných překážek nebo vysokým množstvím robotů v prostoru.

4.3 Předpokládaný čas akcí

Pro všechny akce $a' \in V$ a hrany (a, a') definujeme předpokládaný začátek akce $t'_s(a') = \max\{0; t_s(a'); t'_s(a) + d(a)\}$, kde $d(a)$ značí předpokládanou dobu trvání akce a' a $t_s(a)$ je skutečný čas jejího začátku, pokud akce již proběhla. Předpokládaný konec akce definujeme jako $t'_e(a) = t'_s(a) + d(a)$.

Při exekuci plánu může mezi akcemi dvou různých robotů vzniknout prodleva způsobená zpožděním nebo čekáním jednoho z robotů. Tato prodleva je vlastností hrany typu 2 a nazývá se *slack*. Pro každou hranu $e = (a, w^i_k)$ typu 2 označíme její *slack* $\delta(e) = t'_e(w^i_{k-1}) - t'_e(a)$. Pokud je w^i_k první akce z plánu agenta i, tak $\delta(e) = -t'_e(a)$. *Slack* vrcholu a' značíme $\Delta(a') = \min\{\delta(e)\}$, kde $e = (a, a')$ jsou všechny vstupní hrany typu 2 vrcholu a'.

Pokud je $\delta(e)$ hrany $e = (a, a)'$ záporný, předpokládáme zpoždění akce a' o $-\delta(e)$ kvůli závislosti na akci a. Naopak pro kladné se akce a může ještě zpozdit o $\delta(e)$, aniž by ovlivnila akci a'.

Pro každou akci z plánu má agent *bezpečný interval* ve kterém když akci vykoná, tak nedojde ke kolizi s ostatními roboty. Vývoj bezpečných intervalů v průběhu exekuce sledujeme právě výpočtem *slacku*.

4.4 Metriky exekuce

V případě včas detekované kolize vzniklé kvůli zpoždění jednoho či více robotů lze pomocí přeplánování zabránit zpoždění ostatních robotů. Přeplánování může být výpočetně náročný proces a je tedy důležité, aby bylo využíváno efektivně. K včasné detekci kolizí a k rozhodnutí, zda se vyplatí přeplánovat, lze využít *metriky exekuce*, které monitorují stav exekuce plánu.

4.4.1 Přeplánování v případě čekání robota

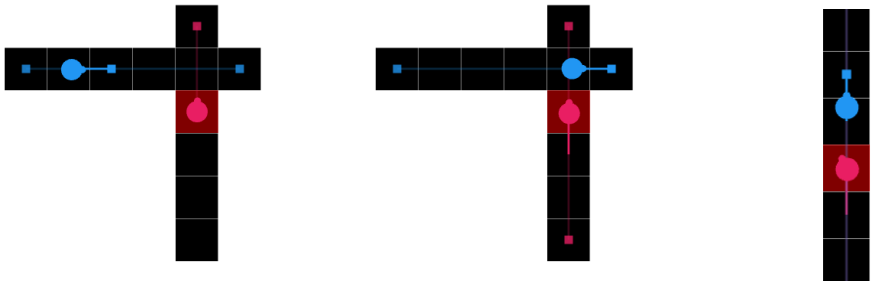
Tato metrika slouží k detekci aktuálních kolizí mezi roboty. K detekci přeplánování dojde v momentě, kdy robot čeká, než jiný robot dokončí akci, aby mohl pokračovat v cestě. Jinak řečeno, pokud existuje nevykonaná akce $a^i_k \in V$ taková, že všechny předchozí akce robota i byly provedeny a zároveň $\Delta(a^i_k) < 0$.

Přeplánování v případě čekání robota může být užitečné v případech, kdy robot čeká na uvolnění křižovatky a přitom robot na kterého čeká je výrazně zpožděn. Tento případ je zobrazen na obrázku 4.2a. Dle plánu má křižovatkou nejprve projet modrý robot a až po něm může červený robot pokračovat v cestě nahoru. Modrý robot je tak zpožděn, že způsobí čekání červeného robota. Další přebytečné čekání červeného robota může být odstraněno přeplánováním, kde po přeplánování křižovatkou nejprve projede červený robot.

Naopak pokud robot měl čekat podle plánu a robot nebo roboti na které čeká jedou podle plánu, tak může být přeplánování zbytečné. Tento případ je zobrazen na obrázku 4.2b. Oba roboti jsou na začátku plánu ve stejné vzdálenosti od křižovatky a aby se předešlo kolizi mezi plány robotů, je

v plánu červeného robota jedna akce čekání. Červený robot tak musí počkat než modrý robot projede i pokud nebude zpožděn.

Na obrázku 4.2c je vyobrazena situace kdy dva roboti jedou za sebou dlouhou chodbou ve které není možné se vyhnout. Zde čekání zajišťuje, aby nedošlo ke kolizi mezi pronásledujícími se roboty. Pokud se zpozdí první robot, musí být zpožděn i druhý robot. Přeplánování neumožňuje tuto situaci vyřešit.



(a) : Čekání robota na křižovatce

(b) : Míjející se roboti

(c) : Roboti jedoucí za sebou

Obrázek 4.2: Příklady detekce přeplánování na základě čekání robota.

4.4.2 Přeplánování na základě hodnoty slacku

Tato metrika detekuje zpoždění robota, které má vliv na další roboty a oproti detekci na základě čekání umožňuje kolizi detekovat ještě než k ní dojde.

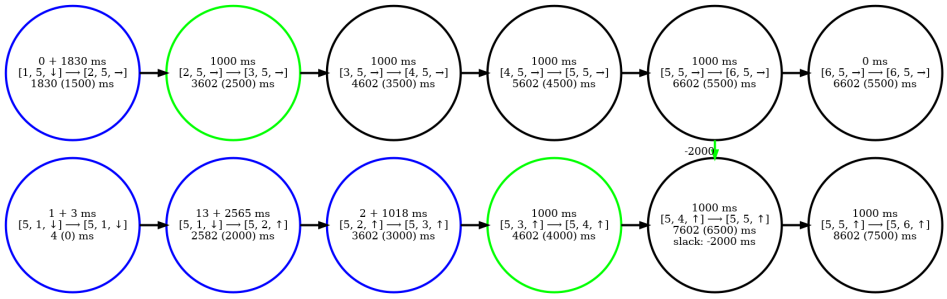
Metrika detekuje potřebu přeplánování, pokud existuje nevykonaná akce z plánu pro kterou je plánovaný *slack* $\Delta_0(a)$ větší než aktuální *slack* $\Delta(a)$ a ten je menší než nula.

Metriku lze ještě rozšířit přidáním parametru. Parametrem je nezáporné číslo n, které určuje jak moc se musí *slack* lišit, aby metrika rozhodla pro přeplánování. Metrika pak detekuje potřebu přeplánování, pokud existuje nevykonaná akce a pro kterou platí $\Delta_0(a) > \Delta(a) + n < 0$.

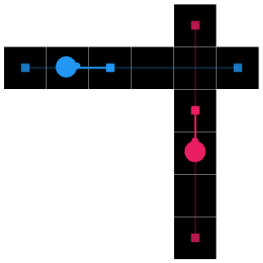
Na obrázku 4.4b jsou dva roboti. Červený robot má cíl označen tečkou vpravo nahoře, modrý má cíl vpravo dole. Červený robot je výrazně zpožděn oproti plánu, podle kterého měl úzkou chodbou projíždět jako první. V případě včasné detekce je možné přeplánovat a nechat modrého robota projet první, díky čemuž roboti dokončí plán dříve, než bez přeplánování.

Na obrázku 4.3 je ADG pro roboty z obrázku 4.4a. Zeleně označené jsou právě vykonávané akce, první řádek grafu jsou akce modrého robota. První informace v každém vrcholu je plánovaná nebo skutečná doba trvání akce, další je počáteční a koncová pozice akce, následuje předpokládaný konec akce a v závorce plánovaný konec akce před začátkem exekuce. Modrý robot měl křižovatkou projíždět jako první a je zpožděn. Předpokládáme, že bude opouštět křižovatku v čase 6602 ms od startu a při tom červený robot má dorazit ke křižovatce v čase 4602 ms od startu, což znamená, že jeho začátek akce se zpozdí o 2000 ms. Plánovaný *slack* akce vjezdu červeného robota

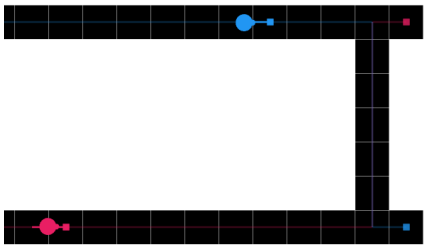
na křižovatku byl -1500 ms. Aktuální se snížil na -2000 ms, proto metrika detekuje potřebu přeplánování a následně bude moct červený robot projet křižovatkou bez potřeby čekat na modrého robota.



Obrázek 4.3: ADG s předpokládanými časy akcí a detekcí přeplánování na základě slacku.



(a) : Roboti jedoucí na křižovatku



(b) : Roboti jedoucí k chodbě, kde se budou míjet

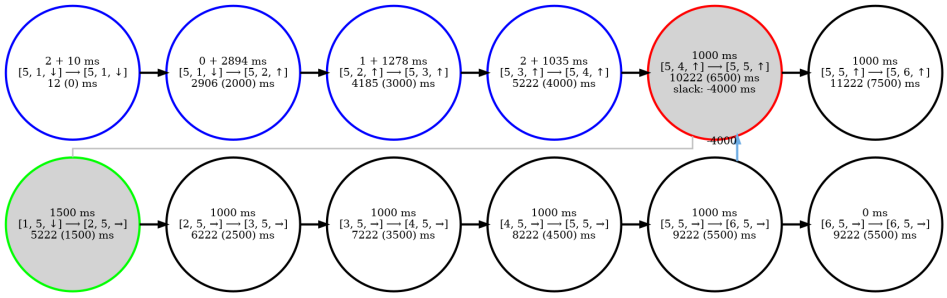
Obrázek 4.4: Příklady detekce přeplánování na základě změn slacku.

4.4.3 Rozdíl mezi počtem vykonaných akcí prvního a posledního robota

Metrika sleduje rozdíl v postupu mezi robotem, který vykonal nejméně akcí a robotem, který vykonal nejvíce akcí. Pokud je rozdíl v počtu dokončených akcí větší než určitý práh, značí to nerovnoměrnost exekuce, která neprobíhá zcela podle plánu a může to indikovat zpoždění jednoho z robotů a potřebu přeplánování.

Nevýhodou této metriky je, že pokud v plánu robota je více akcí čekání za sebou může je robot vykonat ve velmi krátkém čase. Najednou se mu zvýší počet vykonaných akcí, a tím může být výrazně napřed oproti ostatním robotům. Metrika nezohledňuje vzájemné závislosti mezi roboty a může indikovat potřebu přeplánování i v situacích, kdy není potřeba, například v případě kdy cesty robotů jsou na sobě zcela nezávislé.

Na obrázku 4.5 je ADG pro situaci z 4.4a, jen s více zpožděným modrým robotem. Na grafu je vidět, že modrý robot má o tři akce vykonané méně, než červený robot.



Obrázek 4.5: ADG s detekcí přeplánování na základě rozdílu vykonaných akcí.

4.4.4 Zpoždění posledních N akcí robota

Tato metrika sleduje zpoždění v provedení posledních N akcí určitého robota ve srovnání s plánovaným časem jejich provedení. V případě, že robot není schopen dodržet plánované časy provedení akcí, dochází k aktuálnímu zpoždění robota. Může to znamenat, že se aktuálně robot začal zpožďovat. Na rozdíl od sledování absolutního zpoždění vůči plánu umožňuje detekovat situace, kdy robot na začátku jel rychleji, než bylo plánováno a nyní se začal zpomalovat.

Její nevýhoda je stejná jako u předchozí, jelikož zpoždění jednoho robota nemusí mít vliv na zbytek plánu.

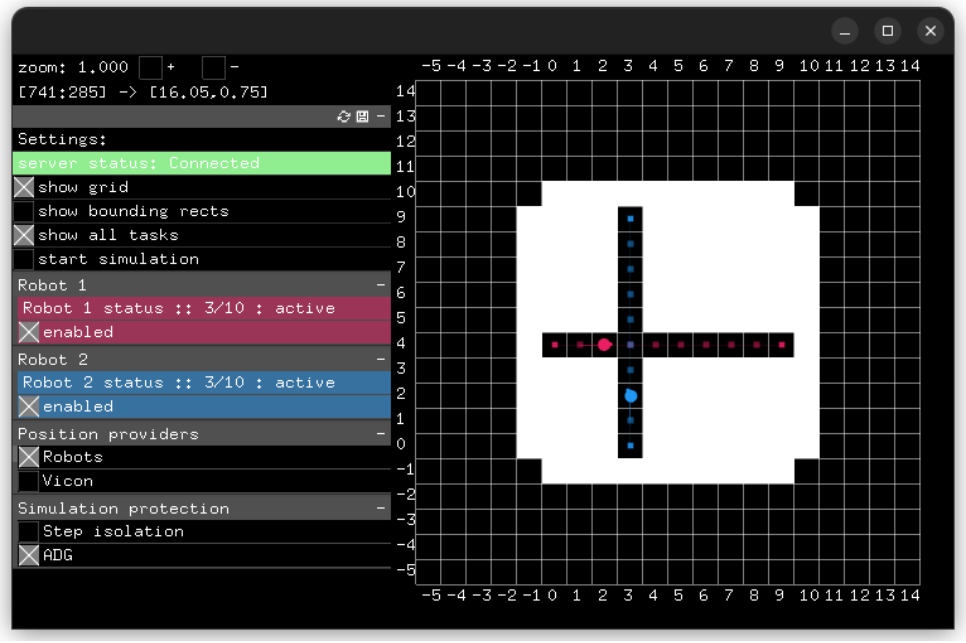
Kapitola 5

Simulátor

Tato kapitola se věnuje Warehouse demonstrátoru vyvíjeném na Českém institutu informatiky, robotiky a kybernetiky (CIIRC), ve kterém byly implementovány a testovány metody z této práci. V sekci 5.1 je popsáno fungování simulátoru a v sekci 5.2 jsou popsány soubory reprezentace prostředí a plánu potřebné ke spuštění simulátoru.

5.1 Běh simulátoru

Simulátor slouží k exekuci plánů pro skupiny robotů. Je možné ho spouštět s reálnými roboty nacházejícími se v budově CIIRC. Má také možnost virtuálního běhu, kde se roboti pouští pouze simulovaně. Na obrázku 5.1 je zobrazen běh simulátoru.



Obrázek 5.1: Ukázka grafického rozhraní simulátoru.

Simulátor umožňuje exekuci buď s využitím ADG nebo pomocí metody step isolation. Metoda step isolation je pro optimalizaci času exekuce za běhu exekuce se zpožděnými roboty nevhodná. Roboti jsou při použití této metody synchronizováni za pomocí časových kroků. Při exekuci je vždy vykonána jedna akce z plánu každého robota zároveň. Než budou agenti moci vykonat další akci plánu, budou muset počkat, než budou dokončeny předchozí akce všech robotů. Není tedy možné brát v potaz jednotlivá zpoždění robotů, jelikož jsou vždy jen v jednom kroku. Další nevýhodou je, že neochraňuje před kolizí v případě *following konfliktu*.

Hlavní část aplikace simulátoru tvoří server, který spravuje exekuci plánu. Server robotům posílá příkazy a přijímá od nich potvrzení o jejich splnění. Kvůli realistickému modelu robotů simulátor počítá i s rotacemi robotů. Robot se tedy musí vždy před vykonáním akce pohybu nejprve natočit směrem akce. Jelikož jsou pro tvorby plánů použity existující veřejně dostupné implementace plánovačů, které zpravidla nepočítají s akcemi rotací, jsou rotace robotů dopočítávány dodatečně z plánu.

5.2 Vstupní soubory pro simulátor

Pro běh simulátoru je třeba znát plán agentů a mapu ve které pohyb probíhá. Tyto informace jsou uložené v textových souborech. Soubory řešení jsou kompatibilní s řešeními vygenerovanými v projektu mapf-IR[8], kde je implementováno několik metod pro řešení problému MAPF.

5.2.1 Soubory map

Mapové soubory jsou uložené v textovém formátu, mohou mít koncovkou `.map` a mají následující strukturu. První čtyři řádky jsou vždy podle vzoru jako v ukázce níže, kde za `height` se nachází výška dané mapy a u `width` její šířka. Zbytek souboru tvoří ASCII reprezentace mapy. Symbol `.` značí přístupnou pozici a písmeno `T` neprůchozí překážku. [13]

```
type octile
height 10
width 10
map
TTT.TTTTTT
TTT.TTTTTT
TTT.TTTTTT
TTT.TTTTTT
..........
TTT.TTTTTT
TTT.TTTTTT
TTT.TTTTTT
TTT.TTTTTT
TTT.TTTTTT
```

5.2.2 Soubory řešení

Soubory řešení obsahují potřebné informace o plánu, mívají koncovku `.sol`. V hlavičce souboru jsou informace o zadání ze kterého bylo řešení vygenerováno, počet agentů, název souboru mapy, startovní a cílové pozice a další metadata plánu. Na konci hlavičky je `solution=`. Za hlavičkou se nachází plán, kde pro každý krok jsou pozice všech agentů.

```
instance=../instances/sample.txt
agents=2
map_file=safe_interval.map
solver=PIBT
solved=1
soc=18
lb_soc=18
makespan=9
lb_makespan=9
comp_time=0
starts=(0,4),(3,0),
goals=(9,4),(3,9),
solution=
0:(0,4),(3,0),
1:(1,4),(3,1),
2:(2,4),(3,2),
3:(3,4),(3,3),
4:(4,4),(3,4),
5:(5,4),(3,5),
6:(6,4),(3,6),
7:(7,4),(3,7),
8:(8,4),(3,8),
9:(9,4),(3,9),
```

Kapitola 6
Implementace

Kapitola se zabývá rozšířením simulátoru o vizualizaci ADG, detekci kolizí a přeplánování. V sekci 6.1 je popsána třída ADG. Sekce 6.2 je o vizualizaci ADG. Sekce 6.3 popisuje výpočet předpokládaných časů akcí, a v sekci 6.4 je popsáno přeplánování cest pro roboty.

6.1 ADG

Implementace ADG je v simulátoru využita pro zajištění bezkolizního průběhu plánu. Tato nová implementace nahrazuje původní, která byla hůře rozšiřitelná o detekci budoucích kolizí. Pro správnou funkčnost ADG předpokládáme, že plán, ze kterého je ADG vytvářen, je platný a neobsahuje *cycle konflikt*.

Při vytvoření instance třídy ADG jsou z plánu MAPF načteny akce agentů, vytvořeny vrcholy ADG a přidány hrany typu 1 a 2 mezi vrcholy. Vrcholy grafu ADG v simulátoru odpovídají akcím v plánu z MAPF plánovače. Následná komunikace serveru s třídou ADG je zajištěna pomocí funkce `canAgentMove` a `markAgentMoveDone`. Funkce `canAgentMove` pro konkrétního robota vrací informaci o tom, zda robot může vykonat další akci. Funkce `markAgentMoveDone` informuje třídu ADG o tom, že robot svou akci dokončil.

6.2 Vizualizace grafu

Obrázky grafu jsou vytvářeny pomocí softwaru Graphviz[1], který je ze souborů zapsaných v programovacím jazyce DOT generuje. Níže je ukázka souboru, který při použití rozvrhovacího algoritmu dot v softwaru Graphviz vygeneruje graf 6.1.

Na prvním řádku souboru je specifikace `digraph`, následovaná jménem grafu, která udává, že se jedná o orientovaný graf. Na řádcích 2-5 jsou nastavovány parametry pro celý graf.

Na šesté řádce začíná `cluster agent_0`, cože je podgraf pro agenta s indexem 0 a definujeme v něm všechny hrany typu 1. Dot kreslí vrcholy grafu tak, aby ty, které jsou součástí jednoho `clusteru`, byly pohromadě.

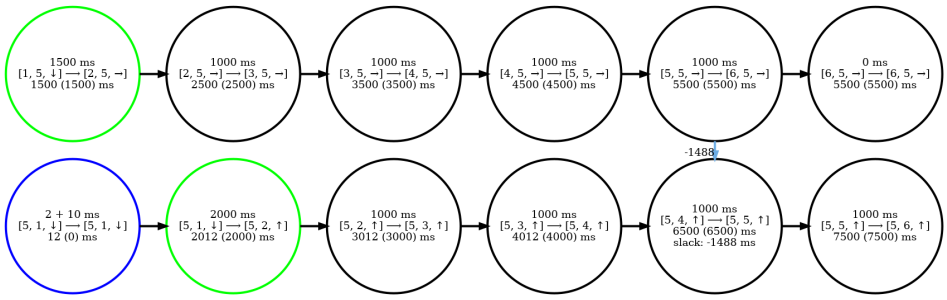
[1] `https://graphviz.org`

Vrchol v grafu je definován a identifikován textovým řetězcem uzavřeným v uvozovkách. Pro jednoduchost a jednoznačnost jsou vrcholy identifikovány dvojicí číslo agenta a pořadí akce. Orientované hrany se vytváří pomocí operátoru `->` mezi jednotlivými vrcholy. Pokud vrcholu nepřidáme popis pomocí parametru `label`, tak bude popisem jeho identifikátor.

Na řádku 24 je vidět nastavení parametrů pro hrany mimo `clustery`. Nastavení parametru `constraint` na `false` zakáže použití dané hrany pro rozvržení pozic vrcholů. Parametr `color` nastavuje barvu hrany. Na řádku 25 je definována jedna hrana typu 2. Na řádcích 26-37 je nastavování barev a popisků k jednotlivým vrcholům.

```
1  digraph Graf {
2      splines="ortho";
3      rankdir="LR";
4      node [shape="circle", fixedsize="true", width=2.5, penwidth
       =3];
5      edge [penwidth=3];
6      subgraph cluster_agent_0 {
7          style="filled";
8          color="white";
9          "0 0" -> "0 1";
10         "0 1" -> "0 2";
11         "0 2" -> "0 3";
12         "0 3" -> "0 4";
13         "0 4" -> "0 5";
14     }
15     subgraph cluster_agent_1 {
16         style="filled";
17         color="white";
18         "1 0" -> "1 1";
19         "1 1" -> "1 2";
20         "1 2" -> "1 3";
21         "1 3" -> "1 4";
22         "1 4" -> "1 5";
23     }
24     edge [constraint="false", color="#6cace4"];
25     "1 4" -> "0 4" [headlabel="-1488"];
26     "0 0" [color="blue"][label="2 + 10 ms\n[5, 1, ↓] ⟶ [5, 1,
       ↓]\n12 (0) ms\n"];

38 }
```



Obrázek 6.1: Ukázka grafu generovaného ze souboru.

6.3 Výpočet předpokládaných časů akcí

Pro možnost predikovat kolize v rámci ADG je třeba znát předpokládané časy akcí. Výpočet předpokládaných časů akcí probíhá podle vzorce v části 4.3. Pro každý vrchol lze určit hodnotu předpokládaného začátku a konce akce jednoznačně, jelikož graf je acyklický.

6.3.1 Algoritmus pro výpočet předpokládaných časů akcí

Pro výpočet předpokládaných časů a pro jejich aktualizaci v průběhu realizace plánu stačí najít topologické očíslování vrcholů, které reprezentují ještě nevykonané akce a v tomto pořadí dopočítat předpokládaný začátek či konec pro každou akci. Předpokládané časy akcí lze dopočítat pomocí algoritmu 2 a jelikož akce s nižším časovým krokem nemohou záviset na akcích s vyšším časovým krokem, lze predikce aktualizovat v tomto pořadí a topologicky seřadit vždy jen akce všech robotů v jednom časovém kroku.

Algoritmus funguje následujícím způsobem. Nejprve je inicializován prázdný zásobník vrcholů ke zpracování (řádek 1). Následně jsou nastaveny všechny vrcholy, které reprezentují již dokončené akce na aktualizované (řádky 2-6), abychom nepočítali předpokládaný čas akce pro akce které už jsou dokončené.

Poté pro všechny neaktualizované vrcholy (řádky 7 a 8) kontrolujeme, zda závisí na jiné akci (řádek 9). Pokud ne, tak aktualizujeme předpokládaný čas akce a označíme vrchol za zpracovaný (řádky 10 a 11). Pokud závisí na jiné akci, tak danou akci přidáme do zásobníku (řádek 13). Následně zpracováváme postupně vrcholy ze zásobníku od posledního přidaného (řádky 14 a 15). Pokud je vrchol již zpracovaný, odstraníme ho ze zásobníku a pokračujeme dále (řádky 17 a 18). Jinak nastavujeme proměnnou `hasDependency` na `false`, což nám značí, že akce w není závislá na jiné akci. Poté procházíme všechny hrany s koncovým vrcholem w a pokud ještě nebyly aktualizované, přidáme je do zásobníku a hodnotu `hasDependency` nastavíme na `true` (řádky 20-23). Pokud vrchol w nebyl závislý na žádné neaktualizované akci (řádek 24), tak ho odstraníme ze zásobníku, aktualizujeme jeho časové odhady a označíme ho za aktualizovaný (řádky 25-27).

6.3.2 Příklad výpočtu předpokládaných časů akcí

Na obrázku 6.2 je ADG s předpokládanými časy dokončení akcí. Každý řádek jsou akce jednoho robota. První informace v každém vrcholu je předpokládaná doba exekuce dané akce. Pro akce pohybu rovně předpokládáme 1000 ms. Pro akce, kde se robot před pohybem musí ještě otočit, připočítáváme 500 ms. Akce čekání ADG nebere v potaz, proto předpokládáme, že jsou vykonány instantně.

Druhé číslo v každém vrcholu představuje odhadovaný čas dokončení dané akce od startu exekuce. Na vrcholu vpravo dole je patrné, že předpokládané čekání robota činí 4500 ms kvůli závislosti na druhém robotovi. Poslední akce

Algoritmus 2 Aktualizace předpokládaných časů akcí

Input: ADG s naměřenými časy akcí
Output: ADG s aktualizovanými předpoklady časů akcí

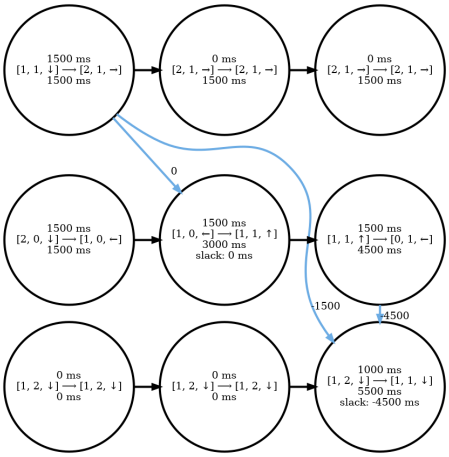
```
1: S ← prázdný zásobník
2: for v ∈ V do
3:    if isFinshed(v) then
4:       isUpdated[v] ← true
5:    else
6:       isUpdated[v] ← false
7: for all v ∈ V do
8:    if not isUpdated[v]  then
9:       if {isUpdated[u] | (u, v) ∈ E} then
10:         updateTimePrediction(v)
11:         isUpdated[v] = true
12:      else
13:         S.push(v)
14:      while not S.empty() do
15:         w ← S.back()
16:         if isUpdated[w] then
17:            S.pop()
18:            continue
19:         hasDependency ← false
20:         for all {u | (u, w) ∈ E} do
21:            if not isUpdated[u] then
22:               S.push(u)
23:               hasDependency ← true
24:         if not hasDependency then
25:            S.pop()
26:            updateTimePrediction(w)
27:            isUpdated[w] ← true
```

v plánu druhého robota má odhadovaný konec 4500 ms od začátku exekuce, zatímco předposlední akce třetího robota má odhadovaný konec 0 ms.

6.4 Přeplánování

Pro vytvoření nového plánu pro přeplánování je využit algoritmus ECBS a jeho implementace z mapf-IR projektu[8].

Při detekci potřeby přeplánování je pozastavena exekuce a vytvořen nový plán z koncových pozic robotů po vykonání poslední zadané akce. Pokud robot akci zrovna vykonává, bude jí muset stejně dokončit, jelikož není možné mu změnit plán v průběhu akce. Následně je vytvořena nová instance třídy ADG s novým plánem. Aby byla zachována návaznost mezi starým a novým plánem, jsou na začátek nového plánu přidány poslední potvrzené pozice robotů, tedy cílové pozice posledních potvrzených akcí. Pokud by tyto akce



Obrázek 6.2: ADG s předpokládanými časy akcí.

do nového plánu přidány nebyly, mohlo by dojít ke kolizi s roboty, kteří ještě dokončují akci z původního plánu.

Detekce přeplánování je na základě metrik v sekci 4.4 a je součástí ADG. Server se periodicky dotazuje ADG, zda by měl vyvolat přeplánování pomocí volání funkce `replan`, která vrací `true` v případě, že by dle zvolené metriky mělo být vyvoláno přeplánování.

Přeplánování v simulátoru má ještě jedno omezení. Přeplánování je vždy povoleno až poté, co všichni roboti dokončí první dvě akce v plánu ADG. Tento přístup zajišťuje simulátoru správný přechod ze starého plánu na nový. Bez tohoto omezení není možné v současné implementaci simulátoru zajistit funkční výměnu plánu. Ačkoliv toto omezení má vliv na přeplánování a experimentální testování metrik, tak jde o rozumný předpoklad, jelikož na začátku nového plánu roboti nemají tolik času se zpozdit, aby bylo potřeba nové přeplánování.

Kapitola 7

Výsledky experimentů

Tato kapitola se věnuje ověření funkčnosti implementované metody ADG a experimentálnímu testování navržených metrik pro detekci kolizí a jejich využití pro vyhodnocení potřeby přeplánování. Testování probíhalo v simulátoru popsaném v kapitole 5.

V sekci 7.1 jsou popsány mapy na kterých probíhalo testování. V sekci 7.2 je testována funkčnost implementace metody ADG. V sekci 7.3 je provedeno měření doby exekuce plánů při použití metrik exekuce pro detekci přeplánování.

7.1 Testovací scénáře

Pro testování máme dvě sady scénářů, některé mapy jsou převzaté z veřejně dostupných MAPF scénářů[1]. V sekci 7.1.1 jsou menší testovací scénáře s konkrétním rozmístěním startů a cílů jednotlivých robotů. V sekci 7.1.2 jsou popsány větší testovací scénáře.

7.1.1 Malé testovací scénáře

Tyto testovací scénáře byly vybrány tak, aby testovaly chování metrik v různých situacích, které mohou nastat během exekuce. Zároveň na nich lze sledovat, zda metriky fungují.

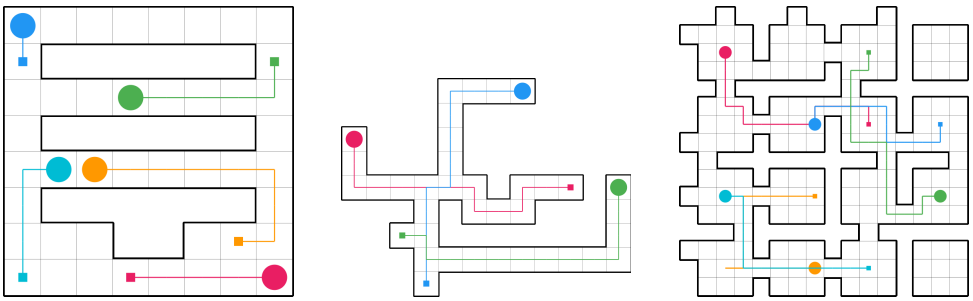
Na obrázku 7.1a je testovací scénář `small_warehouse` pro 5 robotů se vzájemně se nekřížícími cestami. Plán agentů z obrázku je optimální a jelikož jsou cesty robotů na sobě nezávislé, tak není možné pomocí přeplánování zkrátit čas exekuce.

Scénář `corridors` na obrázku 7.1b zobrazuje tři roboty s dvěma křižovatkami. Podle plánu má horní křižovatkou modrý robot projet až po červeném. Spodní křižovatkou má nejprve projet modrý robot a zelený až po něm. Tento scénář ukazuje případ, kdy zpoždění červeného robota může způsobit zpoždění zeleného robota i když se jejich cesty nekříží.

Mapa `room-16-16-4` na obrázku 7.1c představuje několik propojených místností. Ve scénáři jde o dvě skupiny robotů. Skupina vpravo nahoře se

[1] `https://movingai.com/benchmarks/mapf.html`[11]

bude míjet v místnost, kde má cíl červený robot. Podle plánu nejprve místností projede modrý robot, po něm zelený a nakonec červený. Zbylí dva roboti se s předchozí skupinou v tomto plánu nekříží, ale jelikož operují ve společném prostoru, je třeba plánovat jim cesty společně. Plán dvou robotů vlevo dole je takový, že žlutý robot pojede rovně a uvolní cestu druhému robotovi. Poté, co projede, tak žlutý robot může pokračovat v cestě do svého cíle.



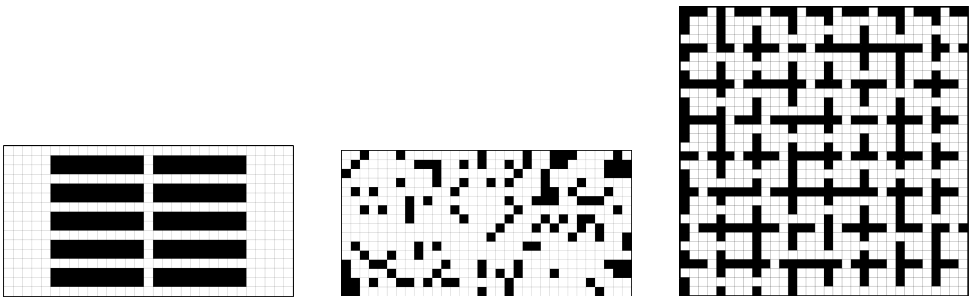
(a) : `small_warehouse` **(b)** : `corridors` **(c)** : `room-16-16-4`

Obrázek 7.1: Menší testovací scénáře.

7.1.2 Větší testovací scénáře

Pro druhou testovací sadu jsou použity mapy na obrázcích 7.2. Scénář tvoří vždy 20 robotů s náhodně vygenerovanými startovními a cílovými pozicemi. Mapa `warehouse` na obrázku 7.2a reprezentuje malé skladiště, kam se na šířku uličky vejde vždy jen jeden robot. Druhá mapa `random-32-16-20` na obrázku 7.2b je mapa s náhodně rozmístěnými překážkami, jde o polovinu mapy `random-32-32-20`. Poslední obrázek 7.2c je celá mapa `room-32-32-4` z veřejných MAPF scénářů.

Tyto testovací mapy byly zvoleny tak, aby byly pro zvolený počet agentů vyřešitelné algoritmem ECBS s mezí suboptimality 1,1. Zároveň byly scénáře navrženy tak, aby byly dostatečně zaplněny agenty, což způsobuje křížení mezi cestami agentů.



(a) : `warehouse` **(b)** : `random-32-16-20` **(c)** : `room-32-32-4`

Obrázek 7.2: Mapy pro testování s větším množstvím robotů.

7.2 Funkčnost metody ADG

Správné fungování nové implementace metody ADG bylo testováno oproti implementaci, která byla v simulátoru původně. Implementace byla testována na výstupy funkce `canAgentMove`. Výstupy obou dvou implementací byly na všech testovaných scénářích stejné. V průběhu všech testování popsaných v této kapitole nebyla detekována žádná přímá kolize mezi simulovanými roboty.

7.3 Testování metrik pro přeplánování

V experimentech porovnáváme dvě hlavní hodnoty, součet dob trvání cest jednotlivých robotů T_{sum} a jejich maximum T_{max}, což je celková doba exekuce.

Plány pro roboty byly vytvářeny pomocí plánovače ECBS s mezí suboptimality 1,1, aby výpočetní doba algoritmu použitého k přeplánování neměla vliv experimenty. Aby bylo možné přímo porovnávat danou metriku považujeme dobu přeplánování za nulovou a reálnou dobu běhu plánovače od doby běhu robotů odečítáme.

Pro řádné otestování metrik je na začátku plánu některým robotům přidáno dodatečné zpoždění.

7.3.1 Nastavení parametrů metrik exekuce

Aby mohly metriky správně fungovat, je třeba u některých z nich vhodně zvolit jejich parametry:

- Metrika přeplánování na základě hodnoty *slacku* má jako parametr číslo n, které určuje jaký musí být minimální rozdíl hodnot *slacku* pro přeplánování. Parametr je blíže popsán v sekci o metrikách (4.4). Hodnota tohoto parametru byla pro potřeby testování nastavena na 1000 ms, což je v simulaci předpokládaná doba trvání akce jízdy rovně. Při této hodnotě by metrika neměla vyhodnotit potřebu přeplánování, pokud podle plánu má robot vjíždět na pozici, ze které jiný robot zrovna odjíždí.

- Metrika přeplánování podle zpoždění posledních N akcí má jako parametr počet posledních akcí, ze kterých počítáme zpoždění a relativní hodnotu, kolikrát může být doba vykonání větší než plánovaná doba. V testech je N nastaveno na 5, aby metrika mohla detekovat zpoždění vzniklé v nedávné době. Potřeba přeplánování je indikována, pokud je doba vykonání akcí minimálně 1,5-krát větší.

- Metrika přeplánování podle rozdílu mezi počtem vykonaných akcí má jako parametr minimální rozdíl počtu vykonaných akcí mezi roboty. Tento parametr je pro testování nastaven na 5. Tento parametr nelze úplně vhodně zvolit pro různorodé scénáře, jelikož u větší množství robotů je více pravděpodobné, že dojde k nerovnoměrné exekuci plánu,

než pro plány s menším množstvím agentů. Lepší výsledky by mohlo mít nastavení parametru v závislosti na počtu robotů.

7.3.2 Výsledky

Každý scénář byl pro každou metriku testován pětkrát, výsledky testování jsou vždy uvedeny v tabulce. Pro každý test byly na začátku plánu zpožděni někteří roboti, zpoždění bylo pro všechny testy z jednoho scénář vždy stejné. Hodnota N je medián z počtu přeplánování. Hodnota T_{max} označuje dobu exekuce a T_{sum} součet dob exekuce všech robotů, u těchto hodnot jsou uvedeny průměrné hodnoty z daných testování. Metriky jsou v tabulce označeny následujícím číslováním:

0. Exekuce bez přeplánování.
1. Přeplánování na základě hodnoty *slacku*.
2. Přeplánování podle zpoždění posledních N akcí.
3. Rozdíl mezi počtem vykonaných akcí prvního a posledního robota.
4. Přeplánování v případě čekání robota.
5. Přeplánování, pokud alespoň polovina metrik vyvolá potřebu přeplánování.

Výsledky testování menších scénářů

Výsledky z testování menších scénářů jsou uvedeny v tabulce 7.1.

Scénář `small_warehouse` byl testován se zpožděním jednoho robota. V tabulce je vidět převažující výsledek, kde metriky nezvolily pro přeplánování. To je v této situaci vhodné řešení.

Ve scénáři `corridors` je na začátku zpožděn červený robot o pět sekund. Z výsledků lze vyčíst, že díky vhodnému přeplánování lze zkrátit celkovou dobu exekuce T_{max} i součet T_{sum}. Nejlepší výsledek na daném scénář měla metrika 5, která na daném scénáři zkrátila součet dob exekucí T_{sum} o 11 %. Nejhorší výsledek měla metrika 3, která nedetekovala potřebu přeplánování. Může to být způsobeno nevhodnou volbou parametru metriky pro tento testovací scénář.

Scénář `room-16-16-4` byl testován pro dvě různé nastavení zpoždění robotů. V obou je zpožděn azurový (cyan) robot a k němu jeden další. V prvním je zpožděn červený robot, což nevede k výrazné změně doby exekuce, jelikož cesta žádného z ostatních robotů na něm není závislá. V druhém je zpožděn modrý robot. Jeho zpoždění vede ke zpoždění celkové exekuce, jelikož na jeho akcích závisí akce zeleného a červeného robota.

Scénář	Metrika	N	T_{max} [s]	T_{sum} [s]
`small_warehouse`	0	0	11,73	41,37
	1	0	12,18	41,31
	2	0	11,45	40,38
	3	0	11,32	40,46
	4	0	11,88	41,51
	5	0	11,39	41,12
`corridors`	0	0	26,65	73,08
	1	1	26,37	66,62
	2	3	26,35	68,10
	3	0	26,76	72,91
	4	3	25,86	66,09
	5	2	**25,72**	**65,19**
`room-16-16-4`, zpoždění červeného robota	0	0	26,54	112,79
	1	1	**25,09**	**108,49**
	2	1	25,82	110,21
	3	1	25,84	112,55
	4	2	25,64	110,59
	5	1	25,34	109,47
`room-16-16-4`, zpoždění modrého robota	0	0	29,67	128,08
	1	1	24,43	114,48
	2	1	29,41	134,26
	3	1	30,34	124,45
	4	3	**24,01**	113,16
	5	1	24,28	**113,13**

Tabulka 7.1: Výsledky měření doby exekuce na menších scénářích.

Výsledky testování větších scénářů

Výsledky z testování větších scénářů jsou uvedeny v tabulce 7.2. Experimenty na každé mapě probíhaly pro dva scénáře, kde každý má jiné náhodné rozmístění robotů a jejich cílů. Na začátku každé exekuce byli zpožděni čtyři roboti z celkového počtu dvaceti o čas v rozmezí pěti až deseti sekund.

Na časech exekuce prvního scénáře mapy `room-32-32-4` je vidět, že v některých případech nemusí být přeplánování vhodné a může prodloužit dobu exekuce. To může být způsobeno využitím suboptimálního plánovače nebo zvolením nevhodných momentů pro přeplánování.

Z výsledků ostatních experimentů je vidět, že se ve většině případů povedlo snížit celkovou dobu exekuce T_{max}. Také můžeme pozorovat, že vysoká hodnota T_{sum} nemusí znamenat prodloužení celkové doby exekuce T_{max} oproti exekuci bez přeplánování. Příkladem toho jsou výsledky experimentů pro metriku 3, kde ve většině experimentů bylo provedeno nejvíce přeplánování a na všech scénářích má nejvyšší průměrnou hodnotu T_{sum}. Přes to byla pomocí přeplánování snížena doba celkové exekuce T_{max}.

Mapa	Metrika	První scénář			Druhý scénář		
		N	T_{max} [s]	T_{sum} [s]	N	T_{max} [s]	T_{sum} [s]
warehouse	0	0	61,90	648,24	0	46,85	**504,62**
	1	3	**57,22**	706,20	3	48,32	590,96
	2	1	57,71	**645,14**	2	**45,64**	586,62
	3	8	57,51	972,16	8	46,81	743,71
	4	5	56,85	781,12	4	46,24	617,98
	5	6	57,68	768,35	5	46,15	648,71
random 32-16-20	0	0	71,92	661,33	0	66,55	**615,11**
	1	2	62,40	**610,60**	2	64,37	645,50
	2	2	**59,76**	716,95	2	67,10	642,37
	3	7	61,92	999,52	10	**64,21**	994,98
	4	7	61,87	856,84	4	66,96	709,00
	5	6	60,72	691,05	4	64,33	693,91
room 32-32-4	0	0	**61,98**	**749,10**	0	68,09	798,72
	1	2	64,76	768,93	2	67,84	**789,24**
	2	1	62,30	750,29	2	67,83	799,51
	3	9	62,24	1013,82	10	**66,49**	1161,31
	4	6	62,20	791,65	6	67,79	858,52
	5	5	63,09	806,63	5	67,92	838,55

Tabulka 7.2: Výsledky měření doby exekuce na větších scénářích.

Shrnutí výsledků

V předchozích experimentech byla ověřena funkčnost metrik pro detekci potřeby přeplánování. Bylo ukázáno, že metriky mohou dobře sloužit ke zmenšení celkové doby exekuce a díky tomu zajistit plynulejší exekuci a rychlejší vykonání plánu. Nebyla nalezena metrika, která by byla jednoznačně lepší než ostatní.

Kapitola 8

Závěr

Cílem této práce byla minimalizace času exekuce plánů v problému MAPF. V práci byla implementována metoda ADG, která byla rozšířena o výpočet předpokládaných časů exekuce jednotlivých akcí. Tohoto rozšíření pak bylo využito k predikci kolizí mezi plány robotů. Pro sledování průběhu plánu byly navrženy a implementovány metriky exekuce.

Tyto metriky byly experimentálně ověřeny na dvou sadách scénářů. První sada obsahovala scénáře s malým počtem robotů navržené tak, aby ověřily fungování metrik. Druhá sada obsahovala scénáře na větších mapách a s větším počtem robotů. Experimenty na první sadě scénářů potvrdily, že využití těchto metrik pro detekci potřeby přeplánování snižuje celkovou dobu exekuce T_{max} i součet dob exekucí T_{sum}. V případě, kdy na sobě cesty robotů nezávisely, metriky nevyvolaly přeplánování. Výsledky experimentů na druhé sadě scénářů ukázaly, že přeplánování může v některých případech zhoršit součet časů exekuce T_{sum}. To může být způsobeno zvolením nevhodného momentu k přeplánování nebo využitím suboptimálního plánovače, který může pro agenty najít horší plán. Celkově z výsledků plyne, že využití metrik navržených v této práci pro indikaci přeplánování snižuje celkovou dobu exekuce ve většině případů.

Ačkoli metriky exekuce prokázaly slibné výsledky při snižování celkové doby exekuce T_{max} ve většině testovaných scénářů, existuje několik oblastí, které by si zasloužily dalšího zkoumání. Použitý přístup k přeplánování byl simulátorem omezen na možnost přeplánování až po prvních dvou akcích vykonaných každým robotem. Jelikož toto omezení může zpozdit nutné přeplánování, jedním z možných vylepšení je možnost úpravy simulátoru, aby bylo možné plán měnit kdykoliv.

Další omezení, které bylo uvažováno v této práci je pozastavení simulace v době přeplánování, které může zpozdit exekuci, pokud bude plánovač moc pomalý. Budoucí práce by se mohla zabývat využitím představených metrik při exekuci, ve které se přeplánování a exekuce překrývají.

Literatura

[1] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner, *Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem*, Proceedings of the International Symposium on Combinatorial Search **5** (2021), no. 1, 19–27 (en).

[2] Alexander Berndt, Niels van Duijkeren, Luigi Palmieri, Alexander Kleiner, and Tamás Keviczky, *Receding horizon re-ordering of multi-agent execution schedules*, IEEE Transactions on Robotics **40** (2023), 1356–1372.

[3] Wolfgang Hoenig, T. K. Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian, and Sven Koenig, *Multi-Agent Path Finding with Kinematic Constraints*, Proceedings of the International Conference on Automated Planning and Scheduling **26** (2016), 477–485 (en).

[4] Wolfgang Hönig, Scott Kiesel, Andrew Tinka, Joseph W. Durham, and Nora Ayanian, *Persistent and Robust Execution of MAPF Schedules in Warehouses*, IEEE Robotics and Automation Letters **4** (2019), no. 2, 1125–1131.

[5] He Jiang, Yulun Zhang, Rishi Veerapaneni, and Jiaoyang Li, *Scaling Lifelong Multi-Agent Path Finding to More Realistic Settings: Research Challenges and Opportunities*, April 2024, arXiv:2404.16162 [cs].

[6] Justin Kottinger, Shaull Almagor, Oren Salzman, and Morteza Lahijanian, *Introducing Delays in Multi-Agent Path Finding*, August 2023, arXiv:2307.11252 [cs].

[7] Hang Ma and Sven Koenig, *AI buzzwords explained: multi-agent path finding (MAPF)*, AI Matters **3** (2017), no. 3, 15–19.

[8] Keisuke Okumura, Yasumasa Tamura, and Xavier Défago, *Iterative refinement for real-time multi-robot path planning*, 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021, `https://kei18.github.io/mapf-IR`, pp. 9690–9697.

[9] Mike Phillips and Maxim Likhachev, *Sipp: Safe interval path planning for dynamic environments*, Proceedings - IEEE International Conference on Robotics and Automation (2011), 5628 – 5635.

[10] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant, *Conflict-based search for optimal multi-agent pathfinding*, Artificial Intelligence **219** (2015), 40–66.

[11] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, and Roman Bartak, *Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks*, June 2019, arXiv:1906.08291 [cs].

[12] Yifan Su, Rishi Veerapaneni, and Jiaoyang Li, *Bidirectional Temporal Plan Graph: Enabling Switchable Passing Orders for More Efficient Multi-Agent Path Finding Plan Execution*, January 2024, arXiv:2401.00315 [cs].

[13] Ondřej Tůma, *The multi-agent path finding demonstrator*, 2022, České vysoké učení technické v Praze.

Seznam zkratek

ADG	Action Dependency Graph
BTPG	Bidirectional Temporal Plan Graph
CBS	Conflict Based Search
CG	Constrained Graph
CIIRC	Český institut informatiky, robotiky a kybernetiky
CT	Constraint Tree
ECBS	Enhanced Conflict Based Search
ICG	Improved Constrained Graph
MAPF	Multi-Agent Pathfinding
SADG	Switchable Action Dependency Graph
SIPP	Safe Interval Path Planning
STN	Simple Temporal Network