FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Multi-agentní hledání cest s více cíli pomocí výrokové splnitelnosti
Student:	Štěpán Tupý
Vedoucí:	prof. RNDr. Pavel Surynek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Multi-agentní hledání cest s více cíli (MG-MAPF) představuje důležité zobecnění problému multi-agentního hledání cest, kdy agenti mají místo jednoho cíle množinu cílů, které je třeba navštívit. V MG-MAPF je tedy třeba určit správné pořadí návštěvy cílů a pro toto pořadí naplánovat bezkonfliktní okružní cesty. Cílem práce je navrhnout zlepšení existujícího algoritmu pro MG-MAPF založeného na převodu úlohy na výrokovou splnitelnost. Jako možné inovace se nabízí například důslednější využití přístupu CEGAR nebo použití řídkých rozhodovacích diagramů. Úkoly pro řešitele jsou následující:

1. Seznámit se s problematikou MAPF a zobecněním s více cíli (MG-MAPF) a souvisejícími řešícími algoritmy založenými na výrokové splnitelnosti.
2. Navrhnout modifikace existujícího algoritmu pro řešení MG-MAPF a implementovat jeho testovací prototyp.
3. Otestovat navržený algoritmus na relevantních benchmarcích.

[1] Pavel Surynek: Multi-Goal Multi-Agent Path Finding via Decoupled and Integrated Goal Vertex Ordering. AAAI 2021: 12409-12417.
[2] Edmund M. Clarke, Anubhav Gupta, Ofer Strichman: SAT-based counterexample-guided abstraction refinement. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 23(7): 1113-1123 (2004).
[3] Guni Sharon, Roni Stern, Ariel Felner, Nathan R. Sturtevant: Conflict-based search for optimal multi-agent pathfinding. Artif. Intell. 219: 40-66 (2015).

Elektronicky schválil/a Ing. Magda Friedjungová, Ph.D. dne 16. února 2023 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

[4] Pavel Surynek: Sparsification for Fast Optimal Multi-Robot Path Planning in Lazy Compilation Schemes. IROS 2021: 7931-7938.

Elektronicky schválil/a Ing. Magda Friedjungová, Ph.D. dne 16. února 2023 v Praze.

Bakalářská práce

MULTI-AGENTNÍ HLEDÁNÍ CEST S VÍCE CÍLI POMOCÍ VÝROKOVÉ SPLNITELNOSTI

Štěpán Tupý

Fakulta informačních technologií
Katedra aplikované matematiky
Vedoucí: prof. RNDr. Pavel Surynek, Ph.D.
11. května 2023

Odkaz na tuto práci: Tupý Štěpán. *Multi-agentní hledání cest s více cíli pomocí výrokové splnitelnosti.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Obsah

Seznam obrázků

Seznam algoritmů

Chtěl bych poděkovat svému vedoucímu prof. RNDr. Pavlu Surynkovi, Ph.D., za vedení této práce, za přínosné konzultace a za jeho pomoc a podporu po celou dobu tvorby práce.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 11. května 2023 Štěpán Tupý

Abstrakt

Tato práce se zabývá hledáním optimálního řešení problému multi-agentního hledání cest s více cíli (MG-MAPF), který je zobecněním multi-agentního hledání cest (MAPF). Úkolem v problému MG-MAPF je nalezení nekonfliktní cesty pro každého agenta z jeho počátečního vrcholu, která pokrývá jemu přiřazenou množinu cílových vrcholů. Přibývá tedy problém volby pořadí, ve kterém agenti své cíle navštíví tak, aby nalezená množina cest byla celkově optimální. Současné algoritmy hledají optimální řešení MG-MAPF pomocí prohledávání stavového prostoru nebo redukcí problému na výrokovou splnitelnost. V této práci jsem implementoval existující algoritmus SMT-Hamiltonian-CBS (SMT-HCBS), který oba přístupy kombinuje. Při redukci MG-MAPF ale vzniká dlouhá booleovská formule a náročné hledání jejího platného ohodnocení snižuje efektivitu tohoto algoritmu. Uvedl jsem tedy dvě nové varianty kódování MG-MAPF do booleovské formule, které využívají méně klauzulí a tím snižují její délku. Redukovaná varianta kódování úspěšně zvýšila efektivitu SMT-HCBS, což se prokázalo při testování na grafech ve tvaru mřížek různé velikosti. SMT-HCBS používající redukované kódování byl na všech testovaných grafech výkonnější než jeho původní varianta.

Klíčová slova multi-agentní hledání cest, MAPF, MAPF s více cíli, MG-MAPF, multi-agentní systémy, výroková splnitelnost, SAT

Abstract

This thesis deals with the search for the optimal solution to the multi-goal multi-agent pathfinding problem (MG-MAPF), which is a generalization of multi-agent pathfinding (MAPF). The task in the MG-MAPF problem is to find a non-conflicting path for each agent starting in its initial position and covering its assigned set of goal vertices. Therefore, the problem of selecting the order in which agents visit their goals so that the set of found paths is overall optimal is introduced. Current algorithms search for optimal MG-MAPF solutions using state space search or reduction of the problem to propositional satisfiability. In this thesis, I implemented an existing algorithm SMT-Hamiltonian-CBS (SMT-HCBS), which combines both approaches. However, the reduction of MG-MAPF produces a long Boolean formula and the difficult search for its valid evaluation decreases the efficiency of this algorithm. I have therefore introduced two new variants of encoding MG-MAPF into the Boolean formula that use fewer clauses and thus decrease the formula's length. The reduced encoding successfully increased the efficiency of SMT-HCBS, which was demonstrated when tested on grid-shaped graphs of different sizes. SMT-HCBS performed better than its original variant on all tested graphs when using the reduced encoding.

Keywords multi-agent pathfinding, MAPF, multi-goal MAPF, MG-MAPF, multi-agent systems, propositional satisfiability, SAT

Seznam zkratek

BFS	Breadth-first search (Prohledávání do šířky)
CBS	Conflict based search
CNF	Conjunctive normal form (Konjunktivní normální forma)
HCBS	Hamiltonian conflict based search
ID	Independence detection
MAPF	Multi-agent pathfinding (Multi-agentní hledání cest)
MDD	Multi-value decision diagram
MG-MAPF	Multi-goal multi-agent pathfinding (Multi-agentní hledání cest s více cíli)
SAT	Boolean satisfiability problem (Problém splnitelnosti booleovské formule)
SMT	Satisfiability modulo theories
SMT-HCBS	SMT-Hamiltonian-CBS
TEG	Time expanded graph (Časová expanze grafu)

Úvod

Multi-agentní hledání cest (MAPF) je fundamentální problém oblasti umělé inteligence. Spočívá v plánování cest pro více agentů v jednom grafu tak, aby se agenti po cestách mohli současně pohybovat a mezi sebou se nesráželi. MAPF slouží jako abstrakce mnoha praktických problémů a má tedy využití například v automatizovaných skladech [1], na letištích [2], v robotice [3] nebo ve videohrách [4]. Na základě potřeb reálného využití se hledá buď optimální nebo suboptimální množina nekonfliktních cest. Kvalita nalezeného plánu může být vyhodnocena podle více různých kritérií, z nichž každé slouží jiným potřebám. Suboptimální řešení jde nalézt rychleji, ale není tak kvalitní, zatímco hledání optimálního řešení je časově mnohem náročnější, ale výsledek je nejlepší možný. Já se budu především zabývat hledáním optimálních řešení, což je výpočetně velmi náročné, protože optimalizace MAPF patří mezi NP-těžké problémy [5].

Algoritmy hledající optimální řešení jsou nejčastěji založené na prohledávání stavového prostoru nebo na redukci MAPF na nějaký jiný problém. Stavový prostor MAPF je ale exponenciálně veliký vůči počtu agentů, takže jeho prohledávání je časově náročné. Algoritmy založené na redukci využívají velmi výkonných řešičů rozsáhle prozkoumaných problémů, jako je například problém splnitelnosti booleovské formule (SAT). SAT řešiče potřebují zadání MAPF zakódované v booleovské formuli. Řešič splnitelnost formule potom vyhodnotí a z jejího ohodnocení je možné extrahovat řešení zakódovaného problému. Tento přístup je méně efektivní na velkých grafech, které generují příliš dlouhou formuli, protože složitost problému SAT roste exponenciálně s počtem proměnných ve formuli.

Klasické MAPF ale není vždy úplně dostačující, takže se nabízí jeho zobecnění, kde každý agent může mít více než jeden cíl (MG-MAPF). Tím přibývá problém určení pořadí návštěv cílů, který je výpočetně náročný i pouze pro jednoho agenta. V této práci se budu soustředit právě na tuto variantu MAPF. Budu navazovat na dřívější výzkum v [6], kde byly navrženy dva algoritmy řešící tento problém. Jeden je založený na prohledávání stavového prostoru a druhý tento přístup kombinuje s redukcí problému na výrokovou splnitelnost a línou konstrukcí formulí.

Cílem mé práce je konkrétně navrhnout a následně implementovat nějakou modifikaci algoritmu *SMT-Hamiltonian-CBS* (SMT-HCBS) [6], která zvýší jeho efektivitu. SMT-HCBS hledá optimální řešení problému MG-MAPF, je založený na výrokové splnitelnosti a k hledání tedy využívá řešiče SAT. V teoretické části nejdříve definuji problém MAPF a jeho variantu s více cíli a potom analyzuji a porovnám existující optimalizační algoritmy, které tyto problémy řeší. V praktické části implementuji algoritmus SMT-HCBS a dále ho modifikuji. Zvýšení jeho efektivity jde dosáhnout například zkrácením booleovské formule, která je předávána řešiči SAT. Můj modifikovaný algoritmus potom otestuji na vhodných benchmarcích a porovnám ho s jeho původní verzí.

Kapitola 1

Multi-agentní hledání cest (MAPF)

Multi-agentní hledání cest je problém teorie grafů, který je zobecněním hledání nejkratší cesty v grafu pro více agentů. Zadáním MAPF je tedy neorientovaný graf, ve kterém jsou umístění agenti, a úkolem je najít cestu pro každého agenta do jemu přiřazeného cílového vrcholu tak, aby se nesrážel s ostatními.

▶ **Definice 1.1.** *Instance MAPF je čtveřice* $\Sigma = (G, A, s_0, g)$*, kde* $G = (V, E)$ *je neorientovaný graf s množinou vrcholů* V *a množinou hran* E*,* $A = a_1, ..., a_k$ *je množina* $k \in \mathbf{N}$ *agentů, kde* $1 < k <= |V|$*,* $s_0 \colon A \mapsto V$ *je funkce označující počáteční konfiguraci agentů v grafu a* $g \colon A \mapsto V$ *je funkce přiřazující každému agentovi vrchol grafu jako jeho cíl.* [7]

Předpokládá se, že čas je diskrétní, takže je rozdělený na jednotlivé časové kroky. V každém kroku se každý agent nachází v nějakém vrcholu grafu. Jeho umístění dohromady s ostatními agenty tvoří v kroku t konfiguraci reprezentovanou funkcí $s_t \colon A \mapsto V$. Mezi časovými kroky může každý agent provést nějakou akci, a to buď počkat ve svém aktuálním vrcholu, nebo se po hraně grafu posunout do sousedního vrcholu. Takto vznikne nová konfigurace agentů v příštím kroku.

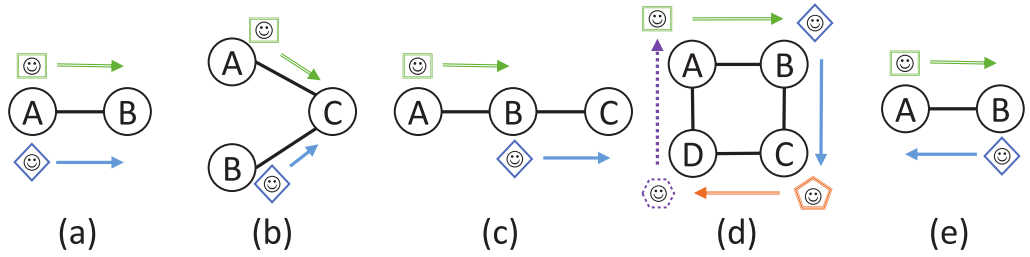
▶ **Definice 1.2.** *Řešení instance MAPF* $\Sigma = (G, A, s_0, g)$ *je posloupnost konfigurací agentů* $S = (s_0, s_1, \ldots, s_{t_M})$ *taková, že všechny konfigurace na sebe navazují pomocí platných akcí. Pro každého agenta* $a_i \in A$ *musí navíc existovat* $t \in 1, 2, \ldots, t_M$ *takové, že platí* $s_t(a_i) = g(a_i)$*.*

Řešení je tedy rozmístění agentů v každém časovém kroku začínající počáteční konfigurací agentů v grafu takové, že agent alespoň jednou navštíví svůj cílový vrchol. Pro agenta a_i lze získat jeho individuální plán jako posloupnost vrcholů $s_0(a_i), s_1(a_i), \ldots, s_{t_M}(a_i)$. Takto vytvořená cesta v MAPF se skládá z posloupnosti akcí začínající v agentově počátečním vrcholu. Řešení je ale platné, pouze pokud v něm neexistují žádné kolize mezi jednotlivými agenty. [8]

1.1 Typy konfliktů

Cílem MAPF je tedy najít takové řešení, aby mezi agenty nevznikaly žádné kolize. Proto je definovaný termín konflikt, který odpovídá srážce dvou nebo více agentů. Řešení MAPF je platné pouze pokud neobsahuje žádné konflikty. Existuje několik druhů konfliktů, které odpovídají různým omezením při praktickém využití.

Vrcholový konflikt Konflikt nastává, pokud se dva agenti nachází ve stejném vrcholu najednou, neboli nastává mezi agenty a_i a a_j pokud v časovém kroku t platí $s_t(a_i) = s_t(a_j)$.



Obrázek 1.1 Ilustrace typů konfliktů v MAPF: (a) hranový konflikt, (b) vrcholový konflikt, (c) konflikt následování, (d) cyklus následování, (e) konflikt výměny [8]

Hranový konflikt Konflikt nastává, pokud dva agenti použijí jednu hranu ve stejnou chvíli a stejném směru, neboli nastává mezi agenty a_i a a_j pokud v časových krocích t a $t+1$ platí $s_t(a_i) = s_t(a_j)$ a zároveň $s_{t+1}(a_i) = s_{t+1}(a_j)$.

Konflikt následování Konflikt nastává, pokud se jeden agent plánuje posunout do vrcholu, který se právě uvolnil. Tento vrchol nebyl dostatečně dlouho volný a v některých situacích by mohlo dojít ke srážce. Formálně tedy nastává mezi agenty a_i a a_j v časových krocích t a $t+1$, pokud platí $s_{t+1}(a_i) = s_t(a_j)$.

Cyklus následování Konflikt nastává, pokud se skupina agentů v jednom časovém kroku plánuje pohnout tak, že se navzájem následují. Tím vznikne několik konfliktů následování, které dohromady tvoří cyklus. Formálně konflikt nastává mezi agenty $a_i, a_{i+1}, \ldots, a_j$ v časových krocích t a $t+1$, pokud platí $s_{t+1}(a_i) = s_t(a_{i+1})$, $s_{t+1}(a_{i+1}) = s_t(a_{i+2})$, … a zároveň $s_{t+1}(a_j) = s_t(a_i)$.

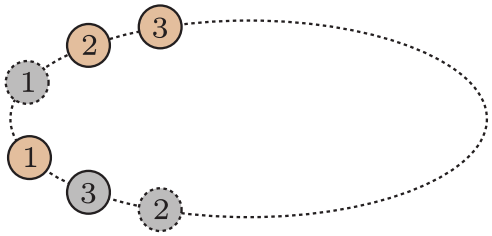
Konflikt výměny Konflikt nastává, pokud dva agenti použijí jednu hranu ve stejnou chvíli, ale každý v opačném směru. Formálně tedy nastává mezi agenty a_i a a_j v časových krocích t a $t+1$, pokud platí $s_t(a_i) = s_{t+1}(a_j)$ a zároveň $s_{t+1}(a_i) = s_t(a_j)$. Těmto konfliktům se někdy také říká hranové konflikty a já je budu v dalších kapitolách této práce také tak nazývat.

Na obrázku 1.1 jsou zobrazeny všechny popsané typy konfliktů. Při výběru, které konflikty by měly být povoleny a které zakázány, zcela záleží na praktickém využití instance MAPF. Pokud nejsou zakázány žádné konflikty, tak se problém rozpadá na soubor problémů hledání nejkratší cesty mezi dvěma vrcholy, který je triviální a řešitelný lineárními algoritmy. Proto v MAPF jsou většinou alespoň nějaké konflikty zakázány, a to přinejmenším hranové konflikty, které jsou nejméně omezující. Z definic konfliktů také vyplývá, že některé z nich jsou podmnožiny jiných. Například zákaz vrcholových konfliktů zakáže implicitně i hranové konflikty a zákaz konfliktů následování zakáže i cykly následování a konflikty výměny.

Výše definované konflikty se vyskytují nejčastěji, ale podle potřeby je možné dodefinovat jakýkoliv jiný druh konfliktu. V definicích MAPF jsou nejčastěji zakázané vrcholové konflikty (a implicitně i hranové). Konflikty výměny bývají někdy povoleny, ale častěji jsou také zakázány a konflikty následování bývají většinou povoleny stejně jako cykly následování. [8]

1.2 Účelové funkce

Jakékoliv řešení MAPF je vyhovující, ale při praktických využitích jsou nějaká řešení lepší než jiná. Na kvalitu řešení lze například nahlížet z hlediska jeho časové náročnosti nebo počtu akcí potřebných k jeho provedení. K popisu kvality řešení se zavádí účelová funkce, která ho ohodnotí, a tato funkce je většinou minimalizována pro optimalizaci řešení. Protože kvalita stejného řešení se může lišit při různých aplikacích, tak se zavádí několik různých účelových funkcí:



Obrázek 1.2 Graf, ve kterém neexistuje řešení MAPF optimalizující makespan i sum of costs zároveň [5]

Makespan Účelová funkce značí počet časových kroků, než každý agent dosáhne svého cíle. Pro řešení $S = (s_0, s_1, \ldots, s_{t_M})$ se tedy makespan rovná t_M. Funkce lze také interpretovat jako délka cesty agenta s nejdelší individuální cestou z počátku do cíle.

Sum of costs Účelová funkce označuje součet akcí všech agentů potřebných k tomu, aby se každý agent dostal do svého cíle. V tomto případě záleží na chování agenta potom, co svého cíle dosáhne. Agent se může chovat dvěma způsoby, a to buď zmizí a uvolní svůj cílový vrchol, nebo ve vrcholu zůstane. Pokud agent zmizí, tak se do sum of costs přičte pouze počet akcí potřebných k dosažení cílového vrcholu, protože dále už v řešení nijak nefiguruje. Pokud ale agent zůstává v grafu, tak nelze vyloučit možnost, že jiný agent se bude potřebovat přesunout do obsazeného cílového vrcholu. V tomto případě musí první agent uhnout a vykonat další akci, která se musí přidat do sum of costs. Pokud se ale agent ze svého cílového vrcholu neplánuje pohnout pryč, tak jeho čekání už celkový součet akcí nezvyšuje. Sum of costs je tedy součet všech akcí, do kterého se nepřičítají čekací akce agentů, kteří se nachází ve svém cílovém vrcholu a už se dále neplánují pohnout jinam.

Fuel Účelová funkce značí celkovou vzdálenost uraženou jednotlivými agenty, a tak symbolizuje množství spotřebovaného paliva, používaného k pohybu. Funkce odpovídá sum of costs, do kterého se nepočítají akce čekání. Díky tomu také odpadá problém určení, který agent ještě plánuje opustit svůj cílový vrchol a který už ne. [8, 9]

Účelové funkce se mohou také kombinovat dohromady tak, že každý agent má přiřazenou svou vlastní odlišnou funkci [10]. V takovém případě se řešení snaží minimalizovat vektor cen cest všech agentů. Také je možné, že mají všichni agenti stejnou účelovou funkci, ale mají přiřazené váhy podle své priority [11].

1.3 Obtížnost

Hledání nejkratší cesty v grafu je problém řešitelný v polynomiálním čase například pomocí algoritmů BFS nebo A*. MAPF se skládá z více takových problémů v jednom grafu, ale kvůli možnosti vzniku kolizí mezi jednotlivými agenty jeho obtížnost výrazně roste. Problém je stále řešitelný v polynomiálním čase [12], ale hledání optimálního řešení je náročnější, protože při jeho hledání se musí procházet stavový prostor, který obsahuje všechny možné konfigurace agentů v grafu. Takových konfigurací je exponenciálně mnoho a neexistuje žádný algoritmus, který by tento stavový prostor dokázal procházet efektivně. V [5] bylo dokázáno, že hledání optimálního řešení MAPF je NP-těžký problém. To platí při všech různých definicích množiny konfliktů, i když nejsou zakázané konflikty cyklického následování, a platí to při minimalizaci jakékoliv účelové funkce z trojice makespanu, sum of costs a fuel.

Také bylo dokázáno, že není vždy možné minimalizovat jakoukoliv z dvojic makespanu, sum of costs a fuel zároveň [5, 13]. Obrázek 1.2 dokazuje toto tvrzení pro dvojici makespanu a sum of costs. Obsahuje instanci MAPF, kde se oranžově označení agenti snaží dostat do svých šedě

označených cílů. Řešení optimalizující makespan je takové, ve kterém se agenti pohybují po směru hodinových ručiček, protože v opačném případě by agent 1 musel obejít skoro celý ovál a makespan zvýšil. V řešení optimalizujícím sum of costs se ale agenti musí pohybovat proti směru hodinových ručiček, protože tak agenti 2 a 3 dříve dorazí do svých cílů a jejich čekání dále sum of costs nezvyšuje.

1.4 Varianty MAPF

Klasická definice MAPF je zčásti flexibilní pro různá praktická využití díky více odlišným typům konfliktů a účelových funkcí. To ale často není dostačující, protože tato definice stále předpokládá, že čas je pouze diskrétní, všechny akce jsou si z pohledu ceny rovny a mnohá další omezení, která nejsou vždy realistická. Proto existují další varianty MAPF, která některá z těchto omezení uvolňují. [8]

1.4.1 MAPF na vážených grafech

Jedno z možných zobecnění MAPF je ohodnocení akcí vahami. Pokud akce ohodnocené nejsou, tak se předpokládá, že všechny zaberou stejný počet časových kroků, což v reálných aplikacích často neplatí. Definice této varianty se oproti klasickému MAPF liší v grafu, ve kterém jsou agenti umístěni. Hrany tohoto grafu jsou vážené podle doby potřebné k přesunu přes ně [14, 15]. Tímto stylem lze MAPF zobecnit do eukleidovského prostoru tak, že vrcholy grafu reprezentují body v prostoru a hrany jejich vzdálenost. [8]

1.4.2 MAPF s pravidly proveditelnosti

V klasickém MAPF je platnost řešení omezená jediným pravidlem, a to neexistencí žádných zakázaných konfliktů. To někdy ale nestačí, takže je možné zavést další pravidla, která musí být splněna, aby řešení bylo platné a proveditelné ve zvoleném prostředí.

Pravidlo robustnosti Často je možné, že se agent zpozdí při vykonávání svých akcí a nebude se stíhat pohybovat podle naplánovaného řešení. K-robustní řešení je naplánované tak, že konflikt nenastane ani když se nějaký agent zpozdí o k časových kroků [16]. Pokud je navíc známá pravděpodobnost zpoždění agenta, tak je možné vytvářet taková řešení, kde je pravděpodobnost srážky agentů shora omezená nějakou hranicí [17].

Formační pravidla Tato pravidla omezují pohyby agenta v závislosti na pozicích ostatních agentů. Taková omezení nepředcházejí pouze kolizím jako klasické MAPF, ale snaží se agenty například udržet v určité formaci [18]. [8]

1.4.3 Plánovaní pohybu

V klasickém MAPF se předpokládá, že se agenti nachází přesně v jednom vrcholu, nemají objem ani tvar a pohybují se konstantní rychlostí. Při plánování pohybu je ale důležité tyto vlastnosti zohledňovat. Každý agent má tedy v této variantě kromě svého umístění ve vrcholu ještě navíc svou konfiguraci popisující jeho vlastnosti relevantní k danému prostředí a zadání.

MAPF s velkými agenty Pokud mají agenti určitý tvar a objem, tak je možné, že v grafu budou kromě vrcholu, ve kterém se nachází, blokovat i další blízké vrcholy nebo hrany [19]. Také je možné, že při pohybu přes hranu budou blokovat jiné hrany, které se s ní protínají nebo se nachází příliš blízko. Je tedy možné zavést nový typ konfliktů, které budou předcházet situacím, kde se dva agenti mohou srazit, přestože se nenachází ve stejném vrcholu nebo hraně, a neporušují tedy základní typy konfliktů [20].

MAPF s kinematickými omezeními Akce MAPF většinou reprezentují pohyb agenta v nějakém směru, a proto je někdy vhodné zavést na nějaké z nich určitá kinematická omezení [21]. Agenti mohou provést tedy jen některé akce v závislosti na své rychlosti a orientaci, na rozdíl od klasického MAPF, kde agent může provést všechny akce dostupné z vrcholu, ve kterém se nachází. [8]

1.4.4 Jiné definice cíle

V praktických aplikacích nemusí mít každý agent přiřazený svůj vlastní cíl, ale může jich mít více než jeden nebo své cíle sdílet s dalšími agenty.

Anonymní MAPF Není vždy důležité, aby byly cíle přiřazené konkrétním agentům. Proto v anonymním MAPF nemá každý agent svůj vlastní cíl, ale všichni dohromady sdílí jednu množinu cílů. Každý agent se potom musí dostat do libovolného cíle nebo se musí každý agent dostat do právě jednoho z cílů, které si mezi sebou mohou jakkoliv rozdělit [22].

Barvené MAPF Tato varianta je zobecnění anonymního MAPF, ve které jsou agenti rozdělení do skupin, které mají svoje množiny cílů. Úkolem agentů je dostat se do libovolného cíle přiřazeného jeho skupině. [23, 24]

Online MAPF V této variantě se řeší posloupnost problémů MAPF, které stále přibývají. Existují dva způsoby, jak mohou další problémy přibývat. Buď agenti dostávají po splnění svého současného úkolu přiřazený nový cíl, do kterého se musí dostat ze své současné pozice, nebo se objevují noví agenti s vlastními cíli a po jejich dosažení zase mizí [25].

První varianta je vhodná například pro automatizované sklady, které jdou ale ještě lépe reprezentovat variantou MAPF nazvanou *multi-agent pickup and delivery* (MAPD) [26], kde agentovi najednou přibydou dva cíle. Jeden z nich reprezentuje místo doručení a druhý reprezentuje místo vyzvednutí dodávky, které musí cestou do cíle navštívit. Druhá varianta reprezentuje například křižovatku, kde přibývají agenti v podobě autonomních vozidel a snaží se jí bezpečně projet. [8]

1.5 MAPF s více cíli (MG-MAPF)

V praktické části této práce se budu především zabývat další variantou MAPF, kde každý agent může mít více cílů. Zadání MG-MAPF je stejné jako zadání klasického MAPF, až na funkci přiřazující agentům jejich cíl. V tomto případě funkce každému agentovi přiřazuje množinu vrcholů místo jediného cíle.

▶ **Definice 1.3.** *Instance MG-MAPF je čtveřice* $\Theta = (G, A, s_0, g)$*, kde* $G = (V, E)$ *je neorientovaný graf s množinou vrcholů* V *a množinou hran* E*,* $A = a_1, ..., a_k$ *je množina* $k \in \mathbf{N}$ *agentů, kde* $1 < k <= |V|$*,* $s_0 \colon A \mapsto V$ *je funkce označující počáteční konfiguraci agentů v grafu a* $g \colon A \mapsto 2^V$ *je funkce přiřazující každému agentovi množinu cílových vrcholů z grafu.*

Úkol každého agenta je vykonat okružní cestu přes všechny své cílové vrcholy. Cíle může navštívit v libovolném pořadí, přibývá tím ale problém určení tohoto pořadí. Řešení má jako při klasickém MAPF podobu posloupnosti konfigurací agentů, ale navíc musí být splněna podmínka návštěvy každého cíle.

▶ **Definice 1.4.** *Řešení instance MG-MAPF* $\Theta = (G, A, s_0, g)$ *je posloupnost konfigurací agentů* $S = (s_0, s_1, \ldots, s_{t_M})$ *taková, že všechny konfigurace na sebe navazují pomocí platných akcí. Pro každého agenta* $a_i \in A$ *a každý jeho cíl* $v_g \in g(a_i)$ *musí navíc existovat* $t \in 1, 2, \ldots, t_M$ *takové, že platí* $s_t(a_i) = v_g$*.*

Stále platí, že řešení je validní, jen pokud neobsahuje žádné konflikty mezi jednotlivými agenty. Také může být vybrána jakákoliv podmnožina konfliktů, které budou zakázány nebo povoleny. Účelové funkce makespan a fuel jsou také definované pořád stejně. Definice sum of costs ale musí být trochu upravena, protože už neexistuje jen jeden cíl, ve kterém může agent čekat, než ostatní agenti dokončí své úkoly. V této práci tedy definuji sum of costs pro MG-MAPF jako součet všech akcí všech agentů provedených předtím, než navštíví všechny své cíle. Agentovy akce se tedy přestávají počítat potom, co navštíví svůj poslední cílový vrchol, a nezáleží na tom, jestli čeká nebo se dále pohybuje. Formálně platí $SoC(S) = \sum_{i=1}^{k} Cena(a_i)$, kde pro cenu cesty jednotlivého agenta platí $Cena(a_i) = \min_{1 \ldots t_M} \{t_c \mid (\forall v_g \in g(a_i))(\exists t \in 1, \ldots, t_c \colon s_t(a_i) = v_g)\}$.

Definici by šlo také upravit tak, že by se agent nejprve musel vrátit do svého počátečního vrcholu nebo do nějakého ze svých cílových vrcholů, kde by zůstal čekat předtím, než by jeho akce dále nezvyšovaly sum of costs. Takovéto změny by ale vyžadovaly jen minimální úpravy řešícího algoritmu.

Najít optimální řešení pro MG-MAPF je stejně jako pro MAPF NP-těžký problém. Na rozdíl od MAPF je ale najít optimální řešení úkolu jednotlivého agenta také NP-těžké. V klasickém MAPF odpovídá úkol jednoho agenta hledání nejkratší cesty mezi dvěma vrcholy v grafu, což je řešitelné v polynomiálním čase. V MG-MAPF ale odpovídá úkol jednoho agenta problému obchodního cestujícího nebo hledání hamiltonovské cesty s nejmenší vahou přes danou podmnožinu vrcholů. [6]

Kapitola 2

Problém splnitelnosti booleovské formule (SAT)

SAT je základní problém oblasti informatiky a matematiky. Jeho cílem je rozhodnout, zda je booleovská formule splnitelná. Je to široce prostudovaný problém, existuje velký počet různých algoritmů, které ho řeší, a navzdory své obtížnosti má důležité aplikace v mnoha různých oblastech. Je využíván například pro plánování a umělou inteligenci, pro verifikaci návrhu obvodů [27] a v kryptografii [28]. Mnoho problémů z těchto oblastí jde polynomiálně redukovat na SAT a pokročilé SAT řešiče dokážou najít jejich řešení. Tímto způsobem jde také hledat optimální řešení grafových problémů jako MAPF a MG-MAPF.

Booleovská formule, jejíž splnitelnost SAT rozhoduje, se skládá z množiny proměnných, logických operátorů a závorek. Po ohodnocení proměnných hodnotami TRUE a FALSE, je výsledná formule buď pravdivá nebo nepravdivá. Cílem SAT je rozhodnout, zda existuje ohodnocení logických proměnných takové, že je celá formule pravdivá. Pokud takové ohodnocení existuje, tak je formule splnitelná, jinak je nesplnitelná. Nejpoužívanější reprezentace booleovské formule je konjunktivní normální forma (CNF).

▶ **Definice 2.1.** *Formule je v CNF, právě když je strukturovaná jako konjunkce klauzulí. Formule se nazývá klauzule, pokud je ve formě disjunkce literálů. Pro každou klauzuli C tedy musí platit $C = l_1 \vee l_2 \vee \cdots \vee l_n$, kde všechny l_i jsou literály. Literál l označuje jeden výskyt logické proměnné x nebo její negace ve formuli, takže platí buď $l = x$ nebo $l = \neg x$. Pro formuli F v CNF tedy platí $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, kde všechny C_j jsou klauzule.*

Většina algoritmů pracuje právě s CNF, takže je nutné booleovské formule nejprve do CNF převést. Každá formule může být převedena do CNF pomocí pravidel booleovské algebry, ale takto zkonstruovaný výsledek může být exponenciálně delší než původní formule. To může negativně ovlivnit efektivitu SAT řešičů, takže je lepší převádět formule jiným způsobem. Například pomocí Tseitinovy transformace [29], která vygeneruje maximálně lineárně více klauzulí, ale navíc vytvoří lineární počet nových proměnných. [30]

Problém SAT je výpočetně složitý a velmi obtížný. Je to první problém, pro který bylo dokázáno, že je NP-úplný [31]. Neexistuje tedy žádný algoritmus, který by ho dokázal vždy efektivně řešit. Jeho složitost je exponenciální vůči počtu proměnných obsažených v booleovské formuli. Konkrétně platí, že časová složitost je v nejhorším případě $O(2^n)$, kde n je počet proměnných ve formuli, protože každá proměnná může být ohodnocena dvěma způsoby a nejhůře musí každý algoritmus vyzkoušet všechny možné kombinace ohodnocení. To znamená, že pro velké instance může být nalezení řešení výpočetně neproveditelné, a i středně velké instance problému mohou být obtížně řešitelné. Varianta 3-SAT, pro kterou platí, že každá klauzule může obsahovat maximálně tři literály, je také NP-úplná. 3-SAT patří stejně jako SAT mezi Karpových 21 NP-úplných

problémů [32] a často se využívá pro důkazy NP-těžkosti jiných problémů. Platí totiž, že pokud je možné polynomiálně redukovat 3-SAT na jiný problém, tak je daný problém minimálně stejně těžký (NP-těžký).

2.1 Řešiče SAT

Navzdory exponenciální složitosti problému SAT existují velmi výkonné a efektivní řešiče, které dokážou řešit instance s velkým množstvím logických proměnných a klauzulí. Byla vyvinuta řada technik a algoritmů pro řešení SAT, které využívají mnoho heuristik a optimalizací. To umožňuje využívat výkonné řešiče SAT pro reálné aplikace v jiných oblastech. SAT řešiče typicky vyžadují booleovskou formuli v CNF a rozhodnou, jestli je splnitelná. Takto je lze využít pro rozhodování o splnitelnosti formulí, které byly vytvořeny zakódováním jiných problémů. Řešiče navíc většinou vrací nalezené ohodnocení splňující booleovskou formuli, ze kterého lze extrahovat zpět řešení kódovaného problému. Existuje mnoho různých řešičů SAT, které mají každý své výhody a obecně platí, že neexistuje žádný řešič, který by byl objektivně lepší na všech možných problémech.

2.1.1 Davis–Putnam–Logemann–Loveland (DPLL)

Algoritmus DPLL [33] je jeden z prvních pokročilých SAT řešičů. Je založený na systematickém prohledávání prostoru ohodnocení logických proměnných. Používá algoritmus *backtracking*, který rekurzivně ohodnocuje jednotlivé proměnné, dokud nenalezne ohodnocení splňující celou formuli. Při každém ohodnocení jedné proměnné se algoritmus rozvětví, protože proměnná může být ohodnocena buď TRUE nebo FALSE. Tímto vzniká prohledávací strom, kterým algoritmus prochází. Pokud tvořené ohodnocení skončí kontradikcí, tak se algoritmus rekurzivně vrátí do úrovně, ve které byla ohodnocena proměnná, která kontradikci tvoří. Na zpáteční cestě se ruší všechna ohodnocení, která na ní byla vytvořena, a na úrovni, kde byl zapříčiněn vznik kontradikce, algoritmus zvolí opačné ohodnocení dané proměnné. Řešiče založené na DPLL využívají různé heuristické funkce například pro volbu pořadí ohodnocování proměnných.

DPLL také používá *jednotkovou propagaci*, která využívá struktury booleovské formule pro její zjednodušení. Pokud v průběhu ohodnocování vznikne klauzule s pouze jedním literálem, který může být ohodnocen kladně, tak je možné jeho proměnnou příslušně ohodnotit a předtím, než algoritmus postoupí dále, se její ohodnocení propaguje do zbytku prohledávacího stromu. Tím mohou vzniknout další jednotkové klauzule, jejichž ohodnocení se také propaguje. Jednotková propagace může výrazně redukovat prohledávaný prostor a dokáže odstranit mnoho redundantních klauzulí. [30]

2.1.2 Conflict-driven clause learning (CDCL)

Mnoho moderních řešičů je odvozeno z algoritmu DPLL a dále ho vylepšují. Jedním z nich je algoritmus CDCL, který analyzuje konflikty mezi ohodnoceními a na jejich základě přidává nové klauzule. Tento přístup se nazývá *clause learning*, protože se tak CDCL učí nové klauzule, které usnadňují hledání ohodnocení splňujícího celou formuli. Když algoritmus detekuje konflikt mezi ohodnocenými proměnných, tak se rekurzivně vrátí stejně jako DPLL, ale navíc přidá novou klauzuli, která nalezený konflikt zakazuje. CDCL obsahuje jednotkovou propagaci stejně jako DPLL, a navíc obsahuje mnoho dalších technik pro řešení problému SAT. Díky tomu je velmi efektivní v praktických aplikacích, takže je hodně využívaný v mnoha různých oblastech. Jedna z prvních implementací CDCL byl řešič *MiniSat* [34], který zavedl další optimalizace pro hledání řešení. Mezi modernější řešiče využívající algoritmus CDCL patří řešič *Glucose* [35], který je na řešiči *MiniSat* založený a zavádí další vylepšení jako například paralelizaci řešení. [30]

Složitost SAT v nejhorším možném případě se odvíjí od počtu logických proměnných, a ne od počtu nebo délky klauzulí obsažených ve formuli. Moderní řešiče ale dokážou často rozhodnout

o splnitelnosti rychleji než v exponenciálním čase, protože využívají mnoho technik, z nichž jsem některé popsal výše. Tyto řešiče jsou výkonnější na určitých typech formulí na základě zvolených optimalizací, které využívají. Například booleovské formule s krátkými klauzulemi obvykle podporují jednotkovou propagaci, protože častěji během řešení vznikají jednotkové klauzule. Krátké klauzule mohou navíc častěji tvořit konflikty mezi ohodnoceními, které se SAT řešiče mohou naučit. Když je booleovská formule celkově kratší, tak řešič dokáže najít ohodnocení splňující formuli rychleji, protože ohodnocení proměnných není omezováno tolika klauzulemi. Navíc mohou řešiče na krátkých formulích rychleji identifikovat konflikty a naučit se nové klauzule. Tyto vlastnosti mohou výrazně ovlivnit výkon SAT řešičů, přestože výsledná formule obsahuje stejný počet logických proměnných. Zvýšení výkonu ale nemusí nastat vždy, protože výkon řešičů závisí na velkém počtu dalších faktorů a celkové struktuře booleovské formule, která nejde jednoduše zachytit.

Kapitola 3

Algoritmy řešící MAPF

Přistupovat k řešení MAPF jde dvěma způsoby: distribuovaně nebo centralizovaně. Při centralizovaném přístupu jsou všichni agenti ovládaní jedním algoritmem, který hledá optimální řešení na základě účelových funkcí definovaných v první kapitole této práce. Naopak v distribuovaném systému má každý agent vlastní výpočetní sílu a jeho komunikace s ostatními agenty může být omezená. Kvůli omezené komunikaci tedy nemusí být vždy možné tímto způsobem najít celkově optimální řešení. Hledání ale zase bývá rychlejší díky tomu, že algoritmy nemusí zohledňovat pohyb všech agentů najednou. Řešící algoritmy jdou dále rozdělit podle kvality řešení, které hledají, a to buď optimální nebo suboptimální. [11]

3.1 Suboptimální řešiče MAPF

Kvůli náročnosti optimalizace MAPF existují řešiče, které nezaručují, že najdou nejlepší nebo vůbec nějaké řešení, ale hledají ho mnohem rychleji. Tyto řešiče jsou efektivnější, protože neprocházejí všechna možná řešení, ale heuristicky vybírají jen některá z nich. Vyplatí se je použít v instancích MAPF s velkým počtem agentů, protože právě v takových případech efektivita optimálních řešičů výrazně klesá. Suboptimální řešiče jdou rozdělit na dvě skupiny podle způsobu hledání řešení. Principy z obou skupin jdou samozřejmě i kombinovat. [11]

Řešiče založené na prohledávání stavového prostoru Řešiče z této kategorie většinou hledají kvalitní řešení, ale v mnoha případech nejsou úplné, takže řešení nemusí vždy najít. Patří mezi ně například algoritmus *Hierarchical cooperative A** (HCA*) [36], který nejprve seřadí agenty a na základě jejich pořadí potom postupně plánuje samostatné cesty. Nalezená individuální cesta je zapsána do rezervační tabulky, která zaznamenává, v jakých vrcholech se agenti budou v určitých časových krocích nacházet. Nové cesty pro další agenty jsou potom plánovány tak, aby nevedly skrz rezervované vrcholy zapsané v tabulce. HCA* má ale několik problémů. Je velmi závislý na pořadí agentů pro hledání cest, které musí být předem určeno. Agenti navíc vždy zůstávají ve svém cílovém vrcholu po jeho dosažení a kvůli rezervaci vrcholů mohou někteří agenti uváznout na místě.

*Windowed-HCA** (WHCA*) [36] je upravená varianta tohoto algoritmu, která jeho problémy částečně řeší. Cesta je plánována pouze na určitý počet kroků napřed a po naplánování tohoto úseku se začne hledat cesta pro dalšího agenta v pořadí. Takovým způsobem má každý agent v určitou chvíli tu nejvyšší prioritu při plánování své cesty, což řeší problém s jejich řazením. Navíc je možné pokračovat v plánování cest i po dosažení cíle, takže cílové vrcholy nezůstávají zablokované.

Řešiče založené na pravidlech pohybu Tyto řešiče neprohledávají stavový prostor, ale plánují cesty na základě specifických pravidel pro pohyb. Dávají přednost úplnosti řešení za určitých podmínek před kvalitou řešení. Při hledání cesty se agenti řídí pouze zavedenými pravidly a vůbec průběžně nekontrolují, jestli jejich naplánovaná cesta netvoří konflikt s nějakou jinou. Například algoritmus *Push and Swap* [37] zavádí pravidlo *push*, které posune agenta k cíli, pokud je to možné, a pravidlo *swap*, které prohodí dva sousední agenty. Tento algoritmus je úplný na grafech, kde jsou vždy alespoň dva vrcholy volné. Další příklad z této kategorie je algoritmus *Tree-based agent swapping strategy* (TASS) [38], který je úplný na stromech.

3.2 Optimální řešiče MAPF

Dále se už v mé práci budu věnovat pouze algoritmům hledajícím optimální řešení MAPF. Neexistuje žádný efektivní algoritmus, který by dokázal najít takové řešení, protože optimalizace MAPF patří mezi NP-těžké úlohy. Optimální řešiče tedy musejí procházet stavový prostor všech různých umístění agentů v grafu, jejichž počet roste exponenciálně s počtem agentů. Dobrý algoritmus k prohledávání stavového prostoru je A*, který vrací optimální řešení při použití přípustné heuristické funkce.

Heuristická funkce v prohledávacích algoritmech odhaduje cenu zbývající cesty do cílového stavu a je přípustná, pokud nikdy neodhadne vyšší cenu, než je cena opravdová. Nejjednodušší taková funkce pro instanci MAPF je součet individuálních heuristik všech agentů. Lepším příkladem je heuristická funkce *sum of individual costs* (SIC), která sčítá opravdovou zbývající vzdálenost agentů do cíle tak, že ignoruje jakékoliv konflikty. Vzdálenost je předem vypočítána mezi každým cílem a každým vrcholem například pomocí BFS spuštěného z každého cílového vrcholu. Při použití SIC minimalizuje A* sum of costs. Pro minimalizaci makespanu lze funkci jednoduše upravit tak, že nepočítá součet ale maximální individuální vzdálenost.

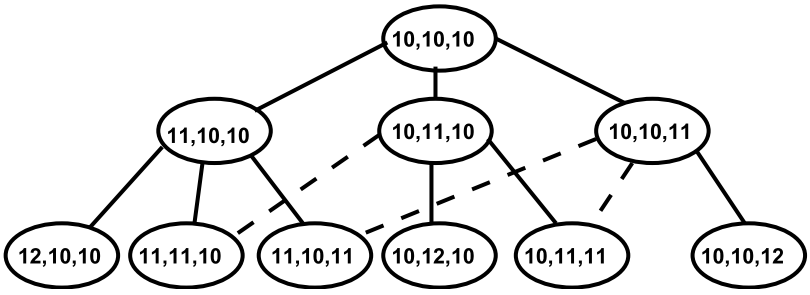
Kvůli velikosti stavového prostoru je ale A* velmi neefektivní. Každý stav může mít až exponenciální počet sousedů vzhledem k počtu agentů, takže při expandování už počátečního stavu může při velkém počtu agentů vzniknout obrovské množství stavů. A* si navíc potřebuje držet všechny stavy v paměti, což není vždy možné. Pokud je v grafu jen málo neobsazených vrcholů, tak sice počet platných stavů (stavy bez konfliktů mezi agenty) může být mnohem menší a neplatné stavy není potřeba držet v paměti, ale stále je nejprve potřeba zjistit, které stavy jsou platné. [11]

3.2.1 Redukce počtu agentů

Hledání optimálního řešení jde zrychlit snížením počtu agentů v problému, což způsobí výraznou redukci stavového prostoru. V [39] byl navržen framework *Independence detection* (ID), který se toho snaží dosáhnout. ID rozděluje agenty do skupin, které reprezentují samostatné instance MAPF. Dvě skupiny jsou navzájem nezávislé, právě když mezi jejich optimálními řešeními nevznikají žádné konflikty. Algoritmus používající ID nejprve umístí každého agenta do jednočlenné skupiny. Potom pomocí A* najde pro každou skupinu individuální řešení a sloučí všechny skupiny, mezi jejichž řešeními se nacházely nějaké konflikty. Pro nově vzniklé skupiny se dále hledá optimální řešení, dokud mezi nimi vznikají konflikty.

S použitím ID klesne složitost optimalizace problému na složitost optimalizace řešení největší skupiny agentů. Algoritmus se tím výrazně zrychlí, protože složitost MAPF je exponenciální vůči počtu agentů. Jelikož je ID framework, tak jde použít i s jinými a výkonnějšími řešícími algoritmy, než je A*, a pomáhá zvýšit jejich efektivitu.

ID jde dále vylepšit přednostním hledáním takových řešení, která nebudou způsobovat nové konflikty mezi skupinami. Toho jde dosáhnout, pokud po vytvoření nové skupiny, preferuje A* při hledání nového řešení takový plán, který tvoří co nejméně konfliktů s ostatními skupinami. Pokud stejně nastane nějaký konflikt, tak je možné přeplánovat obě nalezená skupinová individuální



Obrázek 3.1 Strom vytvořený algoritmem ICTS [11]

řešení tak, aby se skupiny konfliktu vyhnuly. Slučování skupin se ale vždy vyhnout nejde, protože A* stále hledá jen optimální individuální řešení, aby se zachovala celková optimalita.

Další způsob, jak omezit počet agentů v problému, je algoritmus M* [40], který je na A* založený. M* nejprve generuje pouze stavy, ve kterých všichni agenti provádí optimální akci směrem ke svému cíli. Algoritmus takto postupuje, dokud mezi agenty nevznikne konflikt. Potom znovu otevře všechny stavy na cestě zpět k počátečnímu stavu. Pokud bude některý z nich později znovu expandován, tak vygeneruje stavy, ve kterých agenti, kteří způsobili konflikt, provedou už všechny možné akce. Ostatní agenti při generování těchto stavů provedou opět jen optimální akci. [11]

3.2.2 Redukce počtu stavů

Neexistuje algoritmus, který by při prohledávání stavového prostoru expandoval méně stavů než A* se stejnými znalostmi o problému. Toto ale neplatí o počtu vygenerovaných stavů. A* vygeneruje mnoho stavů s hodnotou účelové funkce vyšší než hodnota cílového stavu, které se nikdy neexpandují. Takových stavů může být ve stavovém prostoru MAPF opět exponenciálně mnoho.

Snížení počtu vygenerovaných stavů lze dosáhnout pomocí techniky *Operator decomposition* (OD) [39], která se stavům, které se nikdy neexpandují, snaží vyhýbat. OD k opravdovým stavům navíc vytváří přechodné stavy tak, že nejdříve uvažuje všechny akce pouze jednoho agenta. Z těchto stavů se tvoří další přechodné stavy aplikací stejného postupu na ostatní agenty, dokud nevznikne další opravdový stav. V přechodných stavech jsou povoleny konflikty mezi agenty, protože to nejsou plnohodnotné stavy a agenti se v takové pozici doopravdy nenachází. Při procházení tohoto prostoru ale A* vnímá všechny stavy stejně a objeví cílový stav dříve, než jsou všechny přechodné stavy expandovány. Díky tomu se nevytvoří nadbytečné stavy, které by A* vygeneroval, ale neexpandoval.

Jiný přístup byl zvolen v algoritmu *Enhanced partial expansion A** (EPEA*) [41]. Tento algoritmus při expanzi stavu vygeneruje pouze potomky s nejlepší hodnotou účelové funkce. Potom změní hodnotu účelové funkce právě expandovaného stavu na hodnotu jeho druhého nejlepšího potomka, znovu ho otevře a zařadí do fronty. K vygenerování nejlepších potomků jsou využity heuristické znalosti o doméně, které zajistí, že se vygenerují opravdu jen požadované stavy. Tímto způsobem se stavy generují podle rostoucí hodnoty účelové funkce A* a nevytvoří se žádný stav, který by ji měl vyšší než stav cílový. [11]

3.2.3 Increasing cost tree search (ICTS)

ICTS [42] je dvouúrovňový algoritmus, který hledá optimální řešení MAPF. ICTS na rozdíl od předchozích algoritmů neprohledává stavový prostor všech možných umístění agentů v grafu, ale strom, jehož vrcholy obsahují množinu řešení určité délky. Vrchol stromu obsahuje vektor

$(C_1, \ldots, C_k)$, kde k je počet agentů instance MAPF. Vrchol reprezentuje množinu všech řešení, ve kterých pro každého agenta a_i platí, že jeho individuální cesta má cenu přesně C_i. Vrcholy stromu obsahují i neplatná řešení obsahující konflikty.

Vyšší úroveň algoritmu prohledává tento strom od kořene, který obsahuje vektor s nejkratšími individuálními délkami cest jednotlivých agentů. Potomci každého vrcholu stromu jsou vytvářeni tak, že se jedna z délek cest zvýší o jedna. Tvorba stromu je znázorněna na obrázku 3.1 se stromem, kde mají všichni agenti nejkratší individuální cestu délky deset. Vytvořený strom je postupně procházen pomocí BFS, dokud není nalezen vrchol, který neobsahuje žádné konflikty, reprezentující optimální řešení.

Nižší úroveň je používána k vyhledání odpovídajícího řešení, které neobsahuje žádné konflikty. Je volána v každém vrcholu a hledá pro každého agenta a_i nekonfliktní cesty dlouhé přesně C_i. Pokud žádná taková množina cest neexistuje, vyšší úroveň postoupí do dalšího vrcholu. Vygenerované vrcholy ICTS stromu jdou prořezávat. Pokud neexistuje řešení pro podmnožinu agentů, tak vrchol určitě neobsahuje platné řešení, které se v něm tedy nemusí hledat. [11]

3.3 Conflict based search (CBS)

Stavový prostor MAPF je exponenciálně veliký vzhledem k počtu agentů. Stavový prostor hledání nejkratší cesty v grafu pro jednoho agenta je oproti tomu mnohem menší. Algoritmus CBS, který paří mezi optimální řešiče MAPF, rozkládá MAPF na velké množství takových podproblémů, které jdou řešit v čase lineárním vůči velikosti grafu. CBS začíná hledáním nejkratších individuálních cest a postupně buduje množinu omezení na základě nalezených konfliktů. Při plánování vytváří strom omezení, jehož vrcholy obsahují možná řešení. CBS je dvouúrovňový algoritmus, jehož vyšší úroveň prohledává vytvořený strom a nižší úroveň hledá jednotlivé cesty. Vrchol stromu značený N se skládá z několika hodnot:

Množina omezení ($N.omezení$) Množina obsahuje omezení ve tvaru (a, v, t). Takové omezení znamená, že se agent a nesmí v časovém kroku t nacházet ve vrcholu v. Kořen stromu má tuto množinu prázdnou a každý potomek do množiny přidá právě jedno omezení.

Řešení ($N.řešení$) Množina cest všech agentů, která musí splňovat všechna omezení vrcholu. Může ale mezi jednotlivými cestami obsahovat konflikty, které ještě nebyly zakázány množinou omezení.

Cena řešení ($N.cena$) Cena obsahuje hodnotu zvolené účelové funkce nalezeného řešení. [11]

3.3.1 Průběh algoritmu

Algoritmus 3.1 obsahuje pseudokód **vyšší úrovně** CBS, která má na vstupu instanci MAPF definovanou v první kapitole této práce a vrací optimální řešení ve tvaru množiny cest, které se skládají z akcí pohybu a čekání. Vyšší úroveň prohledává strom omezení od jeho kořene metodou uspořádaného prohledávání (*best-first search*). Kořenový vrchol neobsahuje žádná omezení a jeho řešení se skládá z nejkratších individuálních cest. CBS si udržuje prioritní frontu řazenou podle ceny vrcholů stromu omezení a v každém kroku expanduje první vrchol ve frontě (určený podle ceny jeho řešení).

Po vyjmutí vrcholu z fronty $OPEN$, algoritmus nejprve zkontroluje jeho řešení (řádek 8). Pokud neobsahuje žádné konflikty, tak je to optimální řešení a algoritmus končí (řádek 9). Jinak CBS jeden konflikt vybere (řádek 11) a vytvoří dva syny (řádky 12–17). Konflikt (a_i, a_j, v, t) nastává mezi agenty a_i a a_j ve vrcholu v během časového korku t. Vznik konfliktu znamená, že se alespoň jeden z agentů v tento krok musí vrcholu v vyhnout. Pro optimalitu nalezeného řešení musí CBS vyzkoušet obě možné varianty, takže vytvoří jednoho potomka s přidaným omezením (a_i, v, t) a jednoho s přidaným omezením (a_j, v, t). To způsobí, že v jednom potomkovi současného

Algoritmus 3.1: CBS algoritmus řešící MAPF

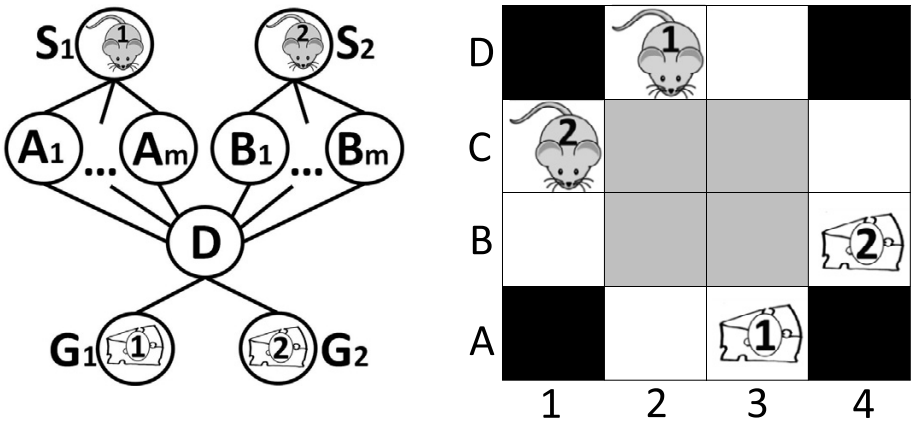
Input: $\Sigma = (G, A, s_0, g)$
1 $R.omezení \leftarrow \emptyset$
2 $R.řešení \leftarrow \{$nejkratší individuální cesta z $s_0(a_i)$ do $g(a_i) \mid a_i \in A\}$
3 $R.cena \leftarrow$ Cena($N.řešení$)
4 vložit R do $OPEN$
5 **while** $OPEN \neq \emptyset$ **do**
6 $N \leftarrow \min(OPEN)$
7 odstranit_min($OPEN$)
8 $konflikt \leftarrow$ validovat($N.řešení$)
9 **if** $konflikt = \emptyset$ **then**
10 **return** $N.řešení$
11 $(a_i, a_j, v, t) \leftarrow konflikt$
12 **foreach** $a \in \{a_i, a_j\}$ **do**
13 $N'.omezení \leftarrow N.omezení \cup \{(a, v, t)\}$
14 $N'.řešení \leftarrow N.řešení$
15 $N'.řešení(a) \leftarrow$ nejkratší cesta z $s_0(a)$ do $g(a)$ respektující $N'.omezení$
16 $N'.cena \leftarrow$ Cena($N'.řešení$)
17 vložit N' do $OPEN$

vrcholu musí změnit svou cestu agent a_i a ve druhém potomkovi ji musí změnit agent a_j. Potom CBS zavoláním nižší úrovně přepočítá nejlepší řešení splňující i přidané omezení, vypočítá cenu nového řešení a potomky zařadí do fronty. Algoritmus nemusí přepočítávat celé řešení, protože nové omezení ovlivní pouze jednoho agenta. Stačí tedy přepočítat pouze jedna cesta. Potom CBS pokračuje s dalším nejlepším vrcholem ve frontě.

Nižší úroveň hledá pro agenta a_i nejkratší cestu z $s_0(a_i)$ do $g(a_i)$ splňující množinu omezení aktuálního vrcholu stromu omezení (řádek 15). Při hledání nemusí brát ohled na ostatní agenty, pouze se musí vyhnout určitým vrcholům v určitých časových korcích. To dokáže jakýkoliv algoritmus pro hledání nejkratší cesty v grafu (například BFS) v lineárním čase. Algoritmus pouze ignoruje stavy zakázané vlivem omezení. Je vhodné, aby nižší úroveň preferovala stavy, které způsobí méně konfliktů s už naplánovanými cestami ostatních agentů, a detekovala duplicitní stavy. Tímto se celková efektivita CBS výrazně zvýší. [11]

3.3.2 Vlastnosti

Algoritmus CBS vždy najde optimální řešení a je úplný, takže řešení najde, pokud nějaké existuje. Nerozpozná ale neexistenci řešení a může se zacyklit. Proto je vhodné řešitelnost instance zkontrolovat před jeho spuštěním například pomocí nějakého úplného suboptimálního řešiče. CBS je v určitých případech efektivnější než A*, který prohledává prostor všech možných konfigurací, protože CBS prohledá méně stavů. To je možné díky tomu, že CBS prohledává trochu jiný stavový prostor. CBS je efektivnější na instancích MAPF, kde existuje zúžení, ve kterém se agenti srážejí, ale při ostatních pohybech mají velkou volnost. Příklad takové instance je levý graf na obrázku 3.2. Pravý graf zase zachycuje instanci MAPF, kde je A* mnohem efektivnější než CBS, protože mezi agenty může ve volném prostoru vznikat mnoho konfliktů a CBS tvoří zbytečně mnoho vrcholů ve svém stromě omezení. Naopak A* bude ignorovat stavy obsahující konflikty, takže relativně rychle dosáhne optimálního řešení. [11]



Obrázek 3.2 Porovnání efektivity CBS a A* na dvou instancích MAPF [11]

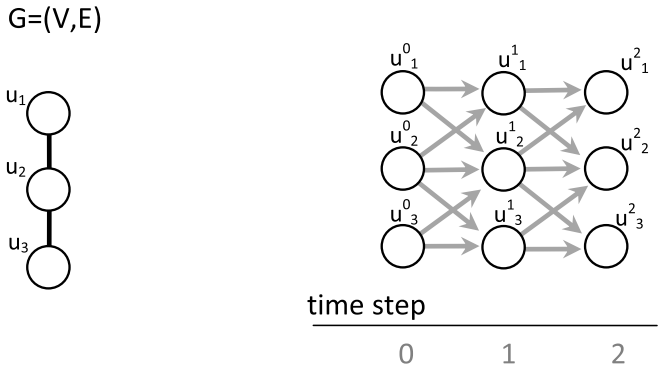
3.3.3 Další možnosti

Pokud CBS detekuje konflikt mezi více než dvěma agenty, tak existují dvě možnosti postupu. Algoritmus buď vytvoří více potomků a každý z nich přidá omezení pro všechny agenty kromě jednoho, nebo se konflikt vnímá pouze jako konflikt mezi prvními dvěma agenty a zbytek se bude řešit až při zpracování potomků. V druhém případě, který je popsán mým pseudokódem, potom budou vznikat duplicitní stavy, které je vhodné detekovat. Časová složitost obou přístupů je podobná.

Výše jsem popsal pouze detekci vrcholových konfliktů, ale stejným způsobem jdou detekovat i další typy. Například hranový konflikt mezi agenty a_i a a_j v hraně $\{v_1, v_2\}$ mezi časovými kroky t a $t+1$ je vyřešen omezením (a_i, v_1, v_2, t), které zakazuje agentovi a_i pohyb z vrcholu v_1 do vrcholu v_2 mezi danými časovými kroky. Podobným způsobem je možné vyřešit i konflikty následování.

CBS lze také použít pro minimalizaci různých účelových funkcí. Stačí vybranou funkci použít ke spočítání ceny nalezeného řešení ve vrcholech stromu omezení. Není nijak potřeba měnit nižší úroveň, protože ta vždy hledá nejkratší povolenou cestu. CBS najde optimální řešení pro všechny přípustné účelové funkce. Účelová funkce je přípustná, pokud nikdy neohodnotí cenu vrcholu výše, než je cena optimálního platného a ve vrcholu povoleného řešení. Funkce makespan, sum of costs i fuel jsou všechny v tomto smyslu přípustné. Při použití účelové funkce fuel už ale není CBS úplný.

CBS je na určitých instancích MAPF velmi neefektivní, jak jsem popsal výše. To platí pro instance, kde mezi určitou skupinou agentů dochází často ke konfliktům. Tento problém je vyřešen použitím algoritmu *Meta-agent CBS* (MA-CBS), který v určitých případech vytvoří z několika agentů jednoho meta agenta. Ten je potom z pohledu vyšší úrovně CBS vnímán jako jeden, přestože obsahuje více normálních agentů. Nižší úroveň musí ale pro meta agenty řešit instance MAPF místo jednoduchého hledání nejkratší cesty. MA-CBS proto musí využívat jiný optimální řešič MAPF, takže ho lze spíše vnímat jako framework nad jinými řešiči. Výhodné je použití algoritmu A*, který je efektivní právě na instancích MAPF, kde CBS efektivní není, a meta agenti přesně takové instance obsahují. MA-CBS funguje tedy podobně jako framework ID, ale neslučuje agenty hned po nalezení prvního konfliktu. Musí detekovat alespoň určitý počet konfliktů mezi dvěma agenty předtím, než je sloučí. Tento počet je udáván parametrem, který může být nastaven v závislosti na podobě řešené instance problému a tím tedy zvyšuje flexibilitu MA-CBS. [11]



Obrázek 3.3 Příklad časové expanze grafu G [7]

3.4 Algoritmy založené na redukci

Řešiče tohoto druhu redukují MAPF na jiný problém, jako například SAT [7, 43, 44, 45], *Integer linear programming* (ILP) [46] nebo *Answer set programming* (ASP) [47]. Tyto problémy jsou rozsáhle prozkoumané a jsou pro ně vyvinuté velmi výkonné a pokročilé řešiče. Tyto řešiče jsou využity pro nalezení řešení redukovaného problému, ze kterého lze získat zpět řešení instance MAPF. Síla existujících řešičů je velká výhoda oproti výše zmíněným algoritmům řešícím MAPF, které nejsou zdaleka tolik prozkoumané a propracované, protože jsou využitelné jen pro jeden konkrétní problém. Na druhou stranu ale řešiče redukovaných problémů nemohou využít žádné heuristické znalosti o konkrétní instanci MAPF, které mohou být využité ve specifickém řešícím algoritmu. Dále se v této práci budu zabývat pouze redukcí na problém SAT. [43]

3.4.1 Kódování do booleovských formulí

Pro použití SAT řešiče je nutné zakódovat instanci MAPF do booleovské formule. Tato formule musí zachytit umístění agentů v grafu, všechna omezení na jejich pohyb, mezi které patři i definice konfliktů, a zároveň konkrétní zadání MAPF. Požadavek pro vytvořenou formuli je, aby byla splnitelná právě tehdy, když existuje řešení kódované instance MAPF. Takových možností, jak zakódovat MAPF, je ale velké množství a je důležité zvolit tu nejvhodnější jak pro rychlost SAT řešiče, tak pro rychlost redukce problému.

K zakódování lze přistupovat pomocí logaritmické nebo přímé reprezentace. Při logaritmické reprezentaci je n-stavová proměnná reprezentována vektorem $\log_2 n$ logických proměnných, které původní proměnné reprezentují stejným způsobem jako jsou čísla reprezentována binárním zápisem. Při přímé reprezentaci je n-stavová proměnná reprezentována vektorem n logických proměnných, ze kterých musí být kladně ohodnocená právě jedna. Při této reprezentaci musí být navíc vytvořeny další formule, které zajistí platnost tohoto omezení. Logaritmický přístup je prostorově efektivnější, protože vytvoří méně proměnných i formulí. Přímý přístup ale podporuje jednotkovou propagaci v SAT řešiči, což urychluje hledání řešení. Není tedy možné jednoduše rozhodnout, který přístup je efektivnější. [43]

Časová expanze grafu G Cesty agentů se skládají z akcí čekání a pohybu, a navíc mohou vést několikrát stejným vrcholem. K zachycení takové cesty není možné použít klasický graf. Proto se zavádí časová expanze grafu (TEG), ve kterém se agenti nacházejí. TEG je acyklický orientovaný graf vytvořený pro fixní makespan. Množina jeho vrcholů obsahuje duplikované vrcholy grafu G v každém možném časovém kroku a množina jeho hran reprezentuje možné akce agentů (i akce čekání), které jsou orientovány pouze ve směru časových kroků. Jeho příklad je vidět na obrázku 3.3, kde je graf G expandován pro makespan roven dvěma.

▶ **Definice 3.1.** *$TEG(G, t_M) = (V, E)$ je časová expanze grafu G do $t_M + 1$ časových kroků, pro jehož množinu vrcholů platí:*

$$V = \{v^t \mid v \in V(G) \wedge t \in 0, 1, \ldots, t_M\}$$

a pro jeho množinu hran platí:

$$E = \{(u^t, v^{t+1}) \mid (\{u, v\} \in E(G) \vee u = v) \wedge t \in 0, 1, \ldots, t_M - 1\}.$$

Hledání řešení instance MAPF s makespanem t_M tedy odpovídá hledání nekonfliktních grafových cest v grafu $TEG(G, t_M)$, jehož vrstvy odpovídají časovým krokům v řešení MAPF. Agenti začínají ve své počáteční konfiguraci v nulté vrstvě grafu a po orientovaných hranách se pohybují do poslední vrstvy. Po cestě musí navštívit svůj cílový vrchol v jakékoliv vrstvě a jelikož je graf acyklický a orientovaný, tak se agenti nachází v právě jednom vrcholu v každém časovém kroku.

Graf časové expanze je použit k zakódování instance MAPF. Vzniká tedy formule $\mathcal{F}(\Sigma, t_M)$, která je splnitelná, právě když instance MAPF Σ má řešení s makespanem t_M. V dřívějším průzkumu bylo otestováno několik různých způsobů kódování MAPF do booleovských formulí a ukázalo se, že nejefektivnější je zakódování pomocí přímé reprezentace. Přestože je toto kódování nejjednodušší a vytvoří nejvíce logických proměnných a formulí, tak jeho struktura, ve které se vyskytuje mnoho krátkých klauzulí, podporuje jednotkovou propagaci v SAT řešičích, což výrazně zvyšuje jejich výkon. [43, 44]

3.4.1.1 Přímé kódování

Zakódování v [43, 44] je představeno pro trochu jiné zadání MAPF, než jsem definoval v této práci, a to že se agenti mohou nacházet ve svém cílovém vrcholu v poslední časový krok řešení, takže ho nemohou pouze navštívit dříve a posunout se jinam. Konkrétní zadání zakazuje vrcholové konflikty a konflikty následování (tedy implicitně i hranové konflikty). V této kapitole popíši tuto verzi zakódování ještě dále zjednodušenou v [7]. Zakódování jde ale jednoduše změnit pro jakoukoliv jinou variantu definice MAPF, například s jinou množinou zakázaných konfliktů.

Předpokládejme instanci MAPF $\Sigma = (G, A, s_0, g)$ a makespan t_M a nechť V značí množinu vrcholů grafu $TEG(\Sigma, t_M)$ a E značí množinu jeho hran. Přímé zakódování vytvoří logickou proměnnou $\mathcal{X}_v^t(a_i)$ pro každého agenta $a_i \in A$ a každý vrchol $v^t \in V$. Tato proměnná je kladně ohodnocená, právě když se agent a_i v časovém kroku t nachází ve vrcholu v. Dále je vytvořena logická proměnná $\mathcal{E}_{u,v}^t(a_i)$ pro každého agenta $a_i \in A$ a každou hranu $(u^t, v^{t+1}) \in E$. Tato proměnná je kladně ohodnocená, právě když se agent a_i přesouvá z vrcholu u do vrcholu v mezi časovými kroky t a $t+1$. Dále jsou uvedeny formule udávající pravidla pro pohyb agentů v grafu:

- Pokud se agent nachází v nějakém vrcholu, tak ho musí opustit pomocí právě jedné hrany. Pro každou zavedenou proměnnou $\mathcal{X}_u^t(a_i)$, kde $t \neq t_M$, jsou vytvořena následující dvě pravidla:

$$\mathcal{X}_u^t(a_i) \implies \bigvee_{(u^t, v^{t+1}) \in E} \mathcal{E}_{u,v}^t(a_i) \tag{3.1}$$

$$\bigwedge_{(u^t, v^{t+1}), (u^t, w^{t+1}) \in E} \neg\mathcal{E}_{u,v}^t(a_i) \vee \neg\mathcal{E}_{u,w}^t(a_i) \tag{3.2}$$

- Pokud agent použije hranu, tak do ní musí vstoupit a zase z ní vystoupit v incidentních vrcholech. Pro každou zavedenou proměnnou $\mathcal{E}_{u,v}^t(a_i)$ je vytvořeno následující pravidlo:

$$\mathcal{E}_{u,v}^t(a_i) \implies \mathcal{X}_u^t(a_i) \wedge \mathcal{X}_v^{t+1}(a_i) \tag{3.3}$$

- Vrchol musí být prázdný předtím, než do něj agent vstoupí (zakazuje konflikty následování). Toto nemusí platit, pokud agent ve vrcholu čeká. Pro každou zavedenou proměnnou $\mathcal{E}^t_{u,v}(a_i)$ je vytvořeno následující pravidlo:

$$\mathcal{E}^t_{u,v}(a_i) \implies \bigwedge_{a_j \in A \wedge i \neq j} \neg \mathcal{X}^t_v(a_j) \tag{3.4}$$

- Žádná dvojice agentů se nesmí zároveň nacházet ve stejném vrcholu (zakazuje vrcholové konflikty). Pro každou zavedenou proměnnou $\mathcal{X}^t_v$ je vytvořeno následující pravidlo:

$$\bigwedge_{a_i, a_j \in A, i \neq j} \neg \mathcal{X}^t_v(a_i) \vee \neg \mathcal{X}^t_v(a_j) \tag{3.5}$$

Předchozí pravidla řídí jen pohyb agentů v grafu. Následující pravidla tvoří zadání instance MAPF:

- Počáteční konfigurace agentů. Pro každou zavedenou proměnnou $\mathcal{X}^0_v(a_i)$ je vytvořeno následující pravidlo:

$$\mathcal{X}^0_v(a_i) \quad \text{pokud } s_0(a_i) = v \tag{3.6}$$
$$\neg \mathcal{X}^0_v(a_i) \quad \text{pokud } s_0(a_i) \neq v \tag{3.7}$$

- Cílové vrcholy agentů. Pro každou zavedenou proměnnou $\mathcal{X}^{t_M}_v(a_i)$ je vytvořeno následující pravidlo:

$$\mathcal{X}^{t_M}_v(a_i) \quad \text{pokud } g(a_i) = v \tag{3.8}$$
$$\neg \mathcal{X}^{t_M}_v(a_i) \quad \text{pokud } g(a_i) \neq v \tag{3.9}$$

Následně je pomocí konjunkce uvedených pravidel vytvořena formule $\mathcal{F}(\Sigma, t_M)$, která obsahuje pouze konjunkce, disjunkce a implikace, takže jde lehce převést do CNF využívané SAT řešiči. Toto specifické zakódování povoluje agentům se vyskytovat ve více vrcholech najednou v jednom časovém kroku. To ale nevadí, protože to nenaruší ekvivalenci mezi splnitelností formule a existencí řešení a taková duplikace jenom ztíží hledání cest ostatních agentů [44]. Nalezené řešení bude stále platné, protože formule nedovoluje agentům z grafu zmizet a v posledním časovém kroku se každý agent musí nacházet pouze ve svém cíli. Agentova cesta tedy určitě povede z jeho počátečního vrcholu do cílového vrcholu a určitě nebude přerušena. Toto zakódování by ale nebylo možné použít, pokud by se agenti nemuseli nacházet ve svém cíli v poslední časový krok řešení.

3.4.2 Průběh hledání řešení

Při hledání řešení minimalizujícího makespan je využit řešič SAT. Řešič rozhodne, jestli je vytvořená formule $\mathcal{F}(\Sigma, t_M)$ splnitelná, a pokud ano, tak správně ohodnotí proměnné. Vytvořená formule je zakódovaná s určitým fixním makespanem, takže řešič vlastně rozhodne, jestli existuje řešení pro daný makespan. Nabízí se tedy začít od nejmenší možné hodnoty makespanu, postupně ji zvyšovat a první nalezené řešení bude řešení s minimálním možným makespanem. Takovéto opakované spouštění dobu běhu algoritmu tolik nezvýší, protože se náročnost problému SAT zvyšuje exponenciálně s počtem logických proměnných, který roste spolu se zvyšováním makespanu. Celkové době běhu bude tedy dominovat doba hledání řešení s nejvyšším zkoušeným makespanem, který je ale optimální a vyzkoušen být musí [43].

Algoritmus 3.2 obsahuje pseudokód popisující optimalizaci MAPF pomocí SAT řešiče. Postupně zvyšuje hodnotu makespanu, dokud řešič SAT nenajde nějaké řešení splňující vytvořenou

formuli (řádek 4). Z vráceného ohodnocení proměnných je potom získáno platné a optimální řešení MAPF (řádek 6). Řešení lze získat pomocí hodnot logických proměnných od počáteční konfigurace agentů po jejich cíle. SAT řešič sice může umístit jednoho agenta do více vrcholů nebo hran najednou, ale pouze jedna pozice bude navazovat na cestu vedoucí od agentova počátečního vrcholu. Hodnoty makespanu se začínají zkoušet až od ceny nejdelší z nejkratších individuálních cest všech agentů (řádek 1), protože to je nejmenší hodnota, pro kterou může nějaké řešení existovat. Tento algoritmus nerozpozná, jestli je instance MAPF řešitelná, takže je vhodné před jeho spuštěním použít nějaký suboptimální (ale úplný) algoritmus k ověření řešitelnosti. [43, 44]

Algoritmus 3.2: Algoritmus založený na SAT minimalizující makespan MAPF

Input: $\Sigma = (G, A, s_0, g)$
1 $t_M \leftarrow \max_{a_i \in A}\{\text{nejkratší individuální cesta z } s_0(a_i) \text{ do } g(a_i) \mid a_i \in A\}$
2 **while** *TRUE* **do**
3 $\mathcal{F}(\Sigma, t_M) \leftarrow$ zakódovat(Σ, t_M)
4 *ohodnoecní* $\leftarrow$ vyřešit_SAT$(\mathcal{F}(\Sigma, t_M))$
5 **if** *ohodnocení* $\neq$ *UNSAT* **then**
6 *řešení* $\leftarrow$ extrahovat_řešení(*ohodnocení*)
7 **return** *řešení*
8 $t_M \leftarrow t_M + 1$

3.5 Algoritmus MDD-SAT

Řešiče založené na redukci na výrokovou splnitelnost jsou méně efektivní na velkých grafech, protože s velikostí grafu roste počet logických proměnných a tím i exponenciálně složitost problému. Další nevýhoda těchto řešičů je komplikované použití jiných účelových funkcí než makespan. Účelové funkce v algoritmech založených na prohledávání stavového prostoru jdou mezi sebou jednoduše měnit, ale redukční algoritmus představený v minulé podkapitole funguje pouze pro makespan. Algoritmus MDD-SAT toto řeší a umožňuje hledat řešení s minimální hodnotou sum of costs pomocí zavedení *cardinality constraint*. MDD-SAT zároveň zvyšuje efektivitu hledání řešení tím, že zmenšuje časovou expanzi grafu a následně vytváří méně logických proměnných i formulí, jejichž splnitelnost je potřeba testovat. [7]

3.5.1 Minimalizace sum of costs

Při minimalizaci sum of costs není možné jednoduše použít stejný přístup jako při minimalizaci makespanu. Je potřeba zavést nějaký nový způsob, jak shora omezit sum of costs, stejně jako je makespan shora omezen počtem kroků časové expanze grafu. Algoritmus MDD-SAT k tomu využívá cardinality constraint [48], jehož pomocí dokáže shora omezit počet kladně ohodnocených proměnných v rámci vybrané množiny.

Pro zakódování instance MAPF je ale stále potřeba časová expanze grafu, která je omezena určitým makespanem. Tento makespan t_M by měl být takový, aby se v grafu $TEG(G, t_M)$ nacházelo každé řešení se sum of costs SoC. Nechť se SoC_0 rovná součtu cen nejkratších individuálních cest všech agentů a omezuje tedy sum of costs zespodu. Nechť t_0 je cena nejdelší z nejkratších individuálních cest všech agentů a slouží jako spodní hranice odhadu makespanu. Nakonec nechť platí $\Delta = SoC - SoC_0$ a nechť Δ tedy značí nadbytečnou cenu řešení nad minimální možnou hodnotou sum of costs.

▶ **Tvrzení 3.2.** *Všechna řešení s hodnotou sum of costs rovnou SoC mají nejvýše makespan t_M pro nějaký makespan $t_M \leq t_0 + \Delta$* [7].

Algoritmus 3.3: Algoritmus založený na SAT minimalizující sum of costs MAPF

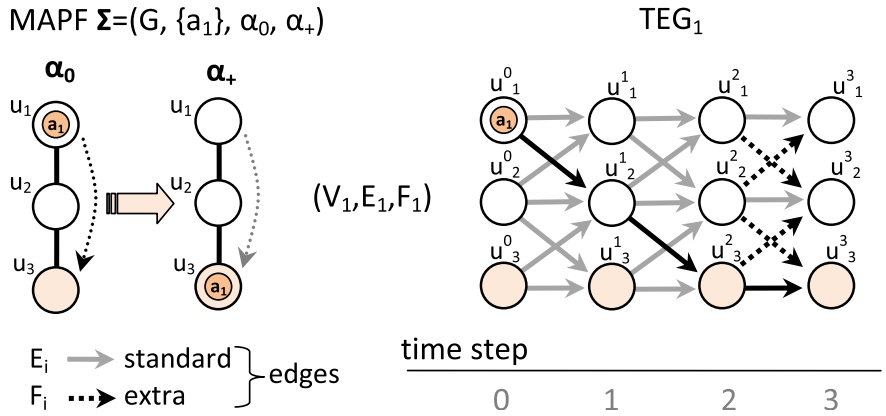
```
   Input: Σ = (G, A, s_0, g)
1  t_0 ← max_{a_i∈A}{nejkratší individuální cesta z s_0(a_i) do g(a_i) | a_i ∈ A}
2  Δ ← 0
3  while TRUE do
4      t_M ← t_0 + Δ
5      F(Σ, t_M, Δ) ← zakódovat(Σ, t_M, Δ)
6      ohodnoecní ← vyřešit_SAT(F(Σ, t_M, Δ))
7      if ohodnocení ≠ UNSAT then
8          řešení ← extrahovat_řešení(ohodnocení)
9          return řešení
10     Δ ← Δ + 1
```

Platí tedy, že časová expanze s makespanem $t_M = t_0 + \Delta$ určitě obsahuje všechna řešení s hodnotou sum of costs rovnou SoC. Algoritmus 3.3 obsahuje pseudokód změněný pro minimalizaci sum of costs a postupuje zhruba stejně jako algoritmus minimalizující makespan. Začíná stejně od minimální možné hodnoty makespanu (řádek 1), ale navíc si udržuje hodnotu Δ značící rozdíl ceny řešení od minimální hodnoty sum of costs (řádek 2). V každé iteraci se dále při zvýšení hodnoty Δ (řádek 10) musí zvýšit i makespan (řádek 4), protože v nejhorším případě nastane konflikt na nejdelší cestě, která se bude muset prodloužit (prodloužit se může maximálně o jednu akci). Poslední rozdíl je zakódování do booleovských formulí na řádku 5, které navíc obsahuje cardinality constraint popsané dále. [7]

3.5.1.1 Cardinality constraint

K nastavení horní hranice sum of costs je nutné namapovat akce agentů na nějaké logické proměnné a na ně aplikovat cardinality constraint, které omezí počet kladně ohodnocených proměnných, a tedy i celkový počet vykonaných akcí. Akce jsou ale reprezentovány hranami v grafu časové expanze, takže pro tento účel je možné využít proměnné $\mathcal{E}^t_{u,v}(a_i)$. Není ale nutné omezovat celkový počet akcí, protože každý agent musí provést alespoň $Cena(a_i)$ akcí, aby dosáhl svého cíle, kde $Cena(a_i)$ značí cenu nejkratší individuální cesty agenta a_i. Omezování celkového počtu akcí je tudíž zbytečné a nebylo by to moc efektivní.

MDD-SAT tedy zavádí nové proměnné pro každého agenta, které označují dodatečné akce nad minimální počet potřebný k dosažení cíle. Agenti mají ale jiné délky nejkratších individuálních



Obrázek 3.4 Časová expanze grafu G pro minimalizaci sum of costs [7]

cest, takže je nutné pro každého agenta a_i zavést vlastní graf časové expanze značený TEG_i. Tento graf obsahuje dvě množiny hran. Množina E_i obsahuje standardní hrany, které začínají ve vrcholech $v^t \in TEG_i$, kde $t < Cena(a_i)$, a všechny hrany označující čekání v cílovém vrcholu $g(a_i)$. Množina F_i obsahuje dodatečné hrany, které začínají ve vrcholech $v^t \in TEG_i$, kde platí $t \geq Cena(a_i)$. Na obrázku 3.4 je časová expanze grafu pro jediného agenta a_1 s oddělenými množinami hran, kde dodatečné hrany jsou označeny tečkovaně.

Pro instanci MAPF $\Sigma = (G, A, s_0, g)$ a makespan t_M je tedy vytvořena množina grafů $TEG_i(\Sigma, t_M)$. Zakódování vytvoří stejné množství logických proměnných $\mathcal{X}_v^t(a_i)$ a $\mathcal{E}_{u,v}^t(a_i)$ jako při minimalizaci makespanu v minulé podkapitole. Navíc je ale vytvořena proměnná $\mathcal{C}^t(a_i)$ pro každého agenta $a_i \in A$ a každý časový krok $t \in 0, 1, \ldots, t_M - 1$, pokud existuje hrana $(u^t, v^{t+1}) \in F_i$. Tato proměnná je kladně ohodnocená, právě když agent vykonal nějakou dodatečnou akci začínající v časovém kroku t. Pro všechny původní proměnné jsou vytvořeny stejné formule jako v minulé podkapitole. Navíc jsou zavedena další dvě nová pravidla omezující sum of costs řešení:

- Pro každého agenta $a_i \in A$ a každou hranu $(u^t, v^{t+1}) \in F_i$ je vytvořeno následující pravidlo:

$$\mathcal{E}_{u,v}^t(a_i) \implies \mathcal{C}^t(a_i) \tag{3.10}$$

- Cardinality constraint, které omezuje hodnotu sum of costs. Toto omezení zajistí, že nalezené řešení použije maximálně Δ (nadbytečná cena řešení) hran ze všech množin F_i:

$$\leq_\Delta \{\mathcal{C}^t(a_i) \mid a_i \in A \wedge t \in 0, 1, \ldots, t_M - 1 \wedge \{(u^t, v^{t+1}) \in F_i\} \neq \emptyset\} \tag{3.11}$$

Cardinality constraint určuje maximální možný počet kladně ohodnocených formulí z uvedené množiny. Pro hranici $\lambda \in \mathbb{N}$ a množinu logických proměnných $X = x_1, x_2, \ldots, x_n$ je cardinality constraint $\leq_\lambda \{x_1, x_2, \ldots, x_n\}$ splněno, právě když je počet kladně ohodnocených proměnných z množiny X menší nebo roven λ. Konjunkcí všech uvedených formulí je vytvořena formule $\mathcal{F}(\Sigma, t_M, \Delta)$. Tato formule je splnitelná, právě když existuje řešení instance MAPF se sum of costs $SoC = SoC_0(\Sigma) + \Delta$, kde $SoC_0(\Sigma)$ označuje součet cen individuálních cest všech agentů. [7]

3.5.2 Multi-value decision diagram (MDD)

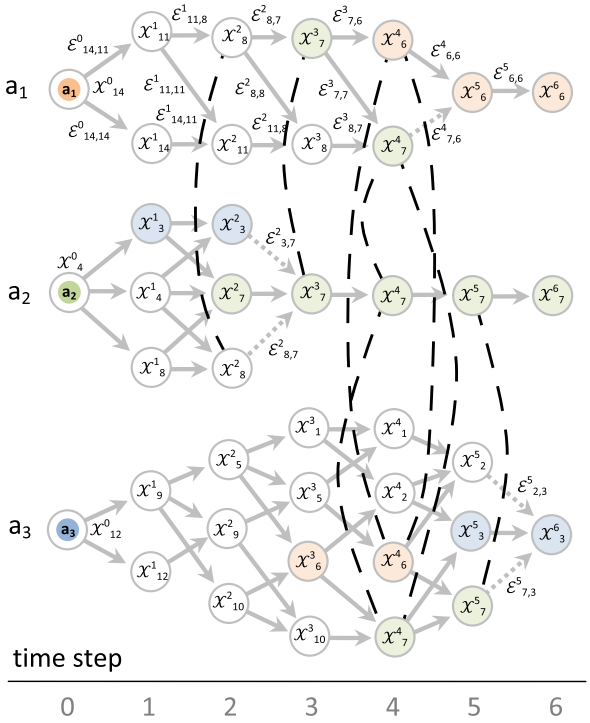
Velikost booleovské formule kódující MAPF má velký efekt na rychlost hledání řešení. Její velikost se odvíjí od počtu logických proměnných, které jsou vytvořeny pro každý vrchol a hranu v časové expanzi TEG. Zmenšením grafu TEG je tedy možné dosáhnout vyšší efektivity řešícího algoritmu. MDD-SAT k tomu využívá graf MDD, který na rozdíl od TEG obsahuje pouze vrcholy dosažitelné z počáteční i cílové pozice agenta. Každý agent je ale v grafu umístěn jinde, takže MDD-SAT pro každého agenta $a_i \in A$ zavádí vlastní MDD_i, který reprezentuje všechny možné cesty z vrcholu $s_0(a_i)$ do cíle $g(a_i)$. Vrchol je v MDD_i tedy obsažen, pouze pokud přes něj nějaká taková cesta vede, takže každý MDD_i obsahuje právě jeden vrchol v prvním časovém kroku, a právě jeden vrchol v posledním časovém kroku.

▶ **Definice 3.3.** *$MDD_i(G, t_M) = (V_i, E_i)$ je acyklický orientovaný graf vytvořený pro agenta a_i z grafu G expanzí do $t_M + 1$ časových kroků. Pro množinu vrcholů grafu $MDD_i(G, t_M)$ platí:*

$$V_i = \{v^t \mid v \in V(G) \wedge t \in 0, 1, \ldots, t_M \wedge Dist(s_0(a_i), v) \leq t \wedge Dist(v, g(a_i)) \leq t_M - t\},$$

kde $Dist(u, v)$ značí vzdálenost mezi vrcholy u a v v grafu G a pro množinu jeho hran platí:

$$E_i = \{(u^t, v^{t+1}) \mid (\{u, v\} \in E(G) \vee u = v) \wedge u^t, v^{t+1} \in V_i\}.$$



Obrázek 3.5 Příklad grafů MDD, jejichž použití výrazně zmenší počet klauzulí nutných k zákazu konfliktů mezi agenty (možné konflikty jsou označeny čárkovanými hranami) [7]

MDD_i nahrazuje TEG_i při tvorbě logických proměnných. Stále platí, že je vytvořena proměnná $\mathcal{X}_v^t(a_i)$ pro každého agenta $a_i \in A$ a každý vrchol $v^t \in V_i$ a proměnná $\mathcal{E}_{u,v}^t(a_i)$ pro každého agenta $a_i \in A$ a každou hranu $(u^t, v^{t+1}) \in E_i$. V_i a E_i jsou ale množiny vrcholů a hran grafů MDD_i, jejichž velikost je redukována oproti stejným množinám grafů TEG_i. Formule reprezentující pravidla a instanci MAPF je také stále generována stejným způsobem, ale je kratší, protože zachycuje pohyby agentů v menším grafu. Může dokonce obsahovat i výrazně méně klauzulí, které zakazují konflikty mezi agenty. Toto je vidět na obrázku 3.5, kde čárkované hrany reprezentují možné konflikty, kterých je mnohem méně, než by jich bylo při použití grafů časové expanze. [7]

3.6 Algoritmus SMT-CBS

Algoritmy založené na prohledávání stavového prostoru i algoritmy založené na redukci na SAT mají své výhody a nevýhody. Prohledávací algoritmy, specificky CBS, jsou efektivnější na grafech, kde nedochází k mnoha konfliktům nebo kde se konflikty vyskytují ve specifických zúženích grafu. Nejsou ale efektivní na hustě obsazených grafech, kde ke kolizím dochází mnohem častěji. Redukční algoritmy, specificky redukující MAPF na SAT, jsou na takových grafech efektivnější, protože nemusí iterovat přes velký počet neplatných stavů. Tyto algoritmy ale zase nejsou efektivní na velkých grafech, k jejichž zakódování potřebují velký počet formulí.

Algoritmus SMT-CBS [49] tyto přístupy kombinuje a získává některé výhody obou typů řešičů. K hledání řešení využívá SAT řešič, od MDD-SAT se ale liší tím, že nevytváří všechny formule najednou, ale líně je konstruuje v průběhu hledání řešení. Tento postup je založen na líném přístupu používaném při řešení problému *satisfiability modulo theories* (SMT). SMT-CBS takto konstruuje menší formule, jejichž výhodou je to, že SAT řešič najde jejich řešení rychleji. Pokud to je ale potřeba, tak SMT-CBS dokáže rychle dotvořit úplnou formuli, která garantuje nalezení platného řešení, pokud takové řešení existuje.

SMT-CBS zavádí neúplnou formuli $\mathcal{H}(\Sigma, t_M)$ vytvořenou stejným postupem jako v algoritmu MDD-SAT. Algoritmus vytváří stejný počet logických proměnných a využívá grafy MDD pro redukci velikosti finální formule. Vynechává ale určitá omezení, která pro instanci MAPF platí. Konkrétně vynechává pravidla zakazující konflikty mezi agenty, takže zajišťuje pouze, že ohodnocení vytvořené formule bude odpovídat cestám propojujícím počáteční vrcholy agentů s jejich cíli. Algoritmus je založený na předpokladu, že takto nalezené řešení může být stále platné a nemusí obsahovat žádné konflikty.

Splnitelnost úplné formule používané v předchozích algoritmech je ekvivalentní s existencí řešení MAPF se zadaným makespanem. Mezi splnitelností neúplné formule a existencí řešení MAPF ale platí pouze implikace. To znamená, že pokud existuje řešení MAPF, tak je formule $\mathcal{H}(\Sigma, t_M)$ určitě splnitelná, ale tato implikace neplatí naopak. V nalezeném ohodnocení neúplné formule se mohou nacházet konflikty, SMT-CBS postupně kóduje nalezené konflikty do formule a líně tedy konstruuje úplnou formuli. Algoritmus takto postupuje, dokud nenalezne platné řešení nebo postupně nevytvoří úplnou formuli obsahující všechna omezení dané instance MAPF. [49]

Algoritmus 3.4: SMT-CBS algoritmus řešící MAPF

```
   Input: Σ = (G, A, s_0, g)
 1 konflikty ← ∅
 2 t_M ← max_{a_i∈A}{nejkratší individuální cesta z s_0(a_i) do g(a_i) | a_i ∈ A}
 3 while TRUE do
 4     (řešení, konflikty) ←SMT-CBS-Fixed(Σ, t_M, konflikty)
 5     if řešení ≠ UNSAT then
 6         return řešení
 7     t_M ← t_M + 1
 8 SMT-CBS-Fixed (Σ, t_M, konflikty)
 9     H(Σ, t_M) ← zakódovat(Σ, t_M, konflikty)
10     while TRUE do
11         ohodnoecní ← vyřešit_SAT(H(Σ, t_M))
12         if ohodnocení ≠ UNSAT then
13             řešení ← extrahovat_řešení(ohodnocení)
14             kolize ← validovat(řešení)
15             if kolize = ∅ then
16                 return (řešení, konflikty)
17             foreach (a_i, a_j, v, t) ∈ kolize do
18                 konflikty ← konflikty ∪ {(a_i, a_j, v, t)}
19                 H(Σ, t_M) ← H(Σ, t_M) ∪ {¬X_v^t(a_i) ∨ ¬X_v^t(a_j)}
20         else
21             return (UNSAT, konflikty)
```

3.6.1 Průběh algoritmu

SMT-CBS funguje na podobném principu jako CBS, a to že začíná s prázdnou množinou omezení zakazující konflikty mezi agenty a během svého běhu ji naplňuje. Na rozdíl od CBS se ale na své vyšší úrovni nevětví, protože jeho nižší úroveň obsahuje řešič SAT, který dokáže vyzkoušet všechny možnosti vyhnutí se nalezeným konfliktům sám. Algoritmus 3.4 obsahuje pseudokód SMT-CBS. Vnější cyklus algoritmu (řádky 3–7) funguje na stejném principu jako předchozí algoritmy založené na redukci na SAT. Postupně iteruje přes zvyšující se hodnoty makespanu,

dokud nenajde nějaké řešení. SMT-CBS ale vytváří neúplnou formuli (řádek 9), kterou dále líně rozšiřuje.

Následuje vnitřní cyklus (řádky 10–21), který hledá řešení pro fixní makespan. Cyklus běží, dokud nenajde platné řešení (řádek 15), nebo nevytvoří nesplnitelnou formuli. V takovém případě, díky implikaci mezi existencí řešení MAPF a splnitelností formule $\mathcal{H}(\Sigma, t_M)$, je jisté, že řešení se zadaným makespanem neexistuje a algoritmus se může posunout k vyšším hodnotám makespanu (řádek 21). Pokud SAT řešič najde řešení splňující formuli, tak SMT-CBS musí nejdřív ověřit, jestli řešení neobsahuje nějaké konflikty, které zatím nebyly přidány do formule. Pokud konflikty neobsahuje, tak je to platné optimální řešení. Jinak se ale musí do formule přidat omezení, která nalezené konflikty zakáží (řádky 17–19). Pseudokód popisuje pouze detekci vrcholových konfliktů, ale stejným způsobem lze pracovat i s ostatními typy konfliktů. SMT-CBS si přenáší množinu nalezených konfliktů mezi všemi iteracemi vnějšího cyklu, protože jinak by SAT řešič opakovaně nacházel ty samé konflikty a tím zbytečně prodlužoval dobu běhu algoritmu. [49]

3.6.2 Vlastnosti

Díky kombinaci algoritmu CBS a redukci MAPF na SAT je algoritmus SMT-CBS výkonný na různých zadáních MAPF. Na malých grafech je stejně výkonný jako MDD-SAT, zatímco efektivita CBS na těchto grafech klesá s těžšími instancemi MAPF s vyšším počtem agentů. Efektivita SMT-CBS je umožněná tím, že algoritmus postupně vytvoří stejnou formuli jako MDD-SAT, ale na cestě k ní se nevětví. To je velká výhoda, protože CBS musí na hustě obsazených grafech s mnoha možnými konflikty iterovat přes velké množství neplatných stavů. Na velkých grafech s lehčími zadáními MAPF s malým počtem agentů se SMT-CBS vyrovná CBS, protože v takových případech tvoří menší formule a SAT řešič najde brzy platné řešení. Na takových instancích MDD-SAT zaostává za CBS, protože tvoří zbytečně velké formule. S větším počtem agentů na velkých grafech, ale efektivita CBS klesá, zatímco SMT-CBS je stále efektivní. SMT-CBS díky línému přístupu pravidelně generuje výrazně méně formulí než MDD-SAT. Takový rozdíl ale nenastává mezi dobami běhu obou algoritmů. To je způsobeno tím, že SMT-CBS musí opakovaně spouštět řešič SAT na neúplných formulích, kde nenajde vždy platné řešení. [49]

Kapitola 4

Algoritmy řešící MG-MAPF

Problém MG-MAPF byl představen teprve nedávno, takže není tolik prozkoumaný jako MAPF a neexistuje mnoho algoritmů, které by ho řešily. Je ale podobný standardnímu MAPF, takže lze k jeho řešení využít principy algoritmů řešících MAPF, které jsem popsal v minulé kapitole. Jediný rozdíl MG-MAPF spočívá v definici funkce přiřazující agentům cíle, která v tomto problému přiřazuje každému agentovi množinu cílových vrcholů, kterými musí alespoň jednou projít. Tím přibývá nová výzva, protože řešící algoritmy musejí naplánovat co nejkratší okružní cesty místo nejkratších cest mezi dvěma vrcholy. V [6] byly uvedeny dva algoritmy hledající optimální řešení MG-MAPF. Jeden z nich je založený na prohledávání stavového prostoru a algoritmu CBS a druhý je založený na redukci MG-MAPF na problém SAT a na algoritmu SMT-CBS.

4.1 Hamiltonian conflict based search (HCBS)

HCBS je algoritmus hledající optimální řešení MG-MAPF. Je odvozen z algoritmu CBS řešícího MAPF a je stejně jako CBS založený na prohledávání stavového prostoru všech možných řešení. Také rozkládá MG-MAPF na více podproblémů sestávajících z hledání individuálních cest a postupně vytváří strom omezení. HCBS je dvouúrovňový algoritmus a s CBS sdílí vyšší úroveň, která prohledává vytvořený strom omezení. Liší se ale v nižší úrovni, ve které CBS hledá pouze nejkratší cesty mezi dvojicemi vrcholů. HCBS musí hledat nejkratší hamiltonovské cesty přes podmnožiny vrcholů grafu. To je exponenciálně složité vůči velikosti daných podmnožin, takže k exponenciálně složitému hledání bezkonfliktního řešení přibývá stejně složitý úkol hledání individuálních cest pro jednotlivé agenty.

▶ **Definice 4.1.** *Hamiltonovská cesta v grafu $G = (V, E)$ začínající ve vrcholu $v \in V$ a pokrývající všechny vrcholy z množiny $U \subseteq V$ je značena $H_P(v, U)$. Je to posloupnost vrcholů $(h_0, h_1, \ldots, h_{t_H})$, pro kterou platí $h_0 = v$, pro každé $t \in 0, 1, \ldots, t_H - 1$ platí $\{h_t, h_{t+1}\} \in E$ a pro každé $u \in U$ existuje $t \in 1, 2, \ldots, t_H$ takové, že platí $u = h_t$. Cena hamiltonovské cesty se rovná počtu jejích hran, takže platí $Cena(H_P(v, U)) = t_H$.*

V rámci problému MG-MAPF se mohou v hamiltonovské cestě vrcholy opakovat, takže to je formálně spíše sled. Klíčová vlastnost HCBS pro hledání takových cest je, že odděluje volbu pořadí návštěv cílů od hledání samotných cest. Toho dosahuje tím, že plánování hamiltonovských cest, které probíhá v nižší úrovni HCBS, je rozděleno na další dvě úrovně. Vyšší z nich pomocí algoritmu A* prohledává prostor všech permutací cílových vrcholů a nejnižší úroveň hledá nejkratší cesty, které cílové vrcholy navzájem propojují v pořadí zvolené permutace. Plánování samotných cest probíhá stejně jako v CBS tak, že se hledané cesty vyhýbají konfliktům určeným množinou omezení z nejvyšší úrovně HCBS. [6]

4.1.1 Průběh algoritmu

Strom omezení algoritmu HCBS obsahuje vrcholy N, které se skládají z hodnot $N.omezení$ obsahující množinu omezení, $N.řešení$ obsahující řešení MG-MAPF, které může obsahovat konflikty mezi agenty, ale nemůže porušit omezení daného vrcholu, a $N.cena$ obsahující hodnotu zvolené účelové funkce nalezeného řešení. HCBS, jehož pseudokód je obsažen v algoritmu 4.1, prohledává tento strom a postupně naplňuje množinu omezení. Kořen stromu je inicializovaný s prázdnou množinou omezení a jeho řešení se skládá z nejkratších individuálních hamiltonovských cest (řádky 1–4).

Vyšší úroveň HCBS prohledává strom omezení stejně jako CBS. Vrcholy stromu jsou prohledávané uspořádaně (*best-first search*) podle své ceny pomocí prioritní fronty $OPEN$. Po vyjmutí vrcholu z fronty je zkontrolováno nalezené řešení (řádek 8), které je optimální, pokud neobsahuje žádné konflikty (řádek 9). Jinak jsou vytvořeni dva synové, kteří se ve svém řešení vyhnou nalezenému konfliktu (řádky 12–18). Pokud je nalezen konflikt (a_i, a_j, v, t), tak si jeden syn do své množiny omezení přidá trojici (a_i, v, t) a druhý syn trojici (a_j, v, t). Omezení (a, v, t) znamená, že se agent a nemůže v časovém kroku t nacházet ve vrcholu v. Tímto způsobem je zajištěno, že jsou vyzkoušeny všechny možné způsoby, jak se mohou agenti vyhnout nalezenému konfliktu.

Nižší úroveň algoritmu hledá nejkratší hamiltonovskou cestu agenta a_i splňující množinu omezení určených vyšší úrovní HCBS. Je pro to využíván algoritmus A* prohledávající stavový prostor všech permutací cílových vrcholů. Jeden stav v tomto prostoru se skládá z hodnot:

Aktuální vrchol ($N.u$) Jeden z cílových vrcholů agenta a_i nebo jeho počáteční vrchol. Tento vrchol je zároveň poslední vrchol částečně nalezené hamiltonovské cesty.

Cesta ($N.cesta$) Částečná hamiltonovská cesta začínající v $s_0(a_i)$ a končící v $N.u$.

Množina dosažených cílových vrcholů ($N.dosažené$) Podmnožina cílových vrcholů agenta a_i pokrytých cestou $N.cesta$.

Vzdálenost od počátku ($N.g$) Vzdálenost mezi počátečním stavem a stavem N, která je rovna délce cesty $N.cesta$.

Hodnota heuristické funkce ($N.h$) Heuristická funkce odhaduje délku zbývající části hamiltonovské cesty.

Počáteční stav prohledávaného prostoru je inicializován vrcholem $s_0(a_i)$, prázdnou množinou dosažených cílů a prázdnou cestou (řádky 20–25). A* postupně tvoří hamiltonovskou cestu tak, že zkouší všechny permutace cílových vrcholů. Stav N obsahuje částečnou cestu, která vede z vrcholu $s_0(a_i)$ do vrcholu $N.u$ a pokrývá všechny cílové vrcholy z množiny $N.dosažené$. V každém kroku je do cesty přidán nový vrchol z cílových vrcholů $g(a_i)$, který ještě nebyl navštíven cestou $N.cesta$ a není tedy v množině $N.dosažené$ (řádky 31–39). Nový vrchol v je napojen pomocí **nejnižší úrovně**, která hledá nejkratší cestu z vrcholu $N.u$ do vrcholu v respektující omezení z nejvyšší úrovně (řádek 32). Taková cesta jde najít například pomocí algoritmu BFS. Aktuální vrchol nového stavu je nastaven na vrchol v (řádek 34), který je také přidán mezi dosažené vrcholy (řádek 36). Stav, jehož množina dosažených vrcholů je stejná jako množina všech cílových vrcholů agenta a_i (řádek 29), je cílový a obsahuje nejkratší hamiltonovskou cestu respektující omezení nejvyšší úrovně HCBS. [6]

A* hledající optimální permutaci cílových vrcholů potřebuje přípustnou heuristickou funkci, která odhadne délku hamiltonovské cesty přes zbývající cílové vrcholy. K tomu jde například využít velikost minimální kostry, která tyto vrcholy pokrývá. Tato definice kostry odpovídá definici minimálního Steinerova stromu. Heuristická funkce využívající velikost tohoto stromu je určitě přípustná, protože nemůže existovat hamiltonovská cesta přes určitou množinu vrcholů, jejíž velikost by byla menší než velikost minimálního Steinerova stromu pokrývajícího stejné vrcholy.

Algoritmus 4.1: HCBS algoritmus řešící MG-MAPF

```
   Input: Θ = (G, A, s_0, g)
 1 R.omezení ← ∅
 2 R.řešení ← {nejkratší hamiltonovská cesta z s_0(a_i) pokrývající g(a_i) | a_i ∈ A}
 3 R.cena ← Cena(R.řešení)
 4 vložit R do OPEN
 5 while OPEN ≠ ∅ do
 6     N ← min(OPEN)
 7     odstranit_min(OPEN)
 8     konflikt ← validovat(N.řešení)
 9     if konflikt = ∅ then
10         return N.řešení
11     (a_i, a_j, v, t) ← konflikt
12     foreach a ∈ {a_i, a_j} do
13         N'.omezení ← N.omezení ∪{(a, v, t)}
14         N'.řešení ← N.řešení
15         N'.řešení(a) ← HCBS-Ordering(Θ, a, N'.omezení)
16         N'.cena ← Cena(N'.řešení)
17         if N'.řešení(a) ≠ Fail then
18             vložit N' do OPEN
19 HCBS-Ordering (Θ, a_i, omezení)
20     R.u ← s_0(a_i)
21     R.cesta ← ()
22     R.dosažené ← ∅
23     R.g ← 0
24     R.h ← Cena(T_S(s_0(a_i) ∪ g(a_i)))
25     vložit R do OPEN
26     while OPEN ≠ ∅ do
27         N ← min(OPEN)
28         odstranit_min(OPEN)
29         if N.dosažené = g(a_i) then
30             return N.řešení
31         foreach v ∈ {g(a_i) \ N.dosažené} do
32             cesta ← nejkratší cesta z N.u do v respektující omezení
33             if cesta ≠ Fail then
34                 N'.u ← v
35                 N'.cesta ← N.cesta · cesta
36                 N'.dosažené ← N.dosažené ∪{v}
37                 N'.g ← N.g + Cena(cesta)
38                 N'.h ← Cena(T_S(N'.u ∪ {g(a_i) \ N'.dosažené}))
39                 vložit N' do OPEN
40     return Fail
```

▶ **Definice 4.2.** *Steinerův strom v grafu $G = (V, E)$ pokrývající vrcholy z množiny $U \subseteq V$ je značen $T_S(U) = (V_U, E_U)$. Je to strom, pro jehož množinu vrcholů platí $U \subseteq V_U \subseteq V$ a pro množinu jeho hran platí $E_U \subseteq E$. Cena Steinerova stromu je rovna počtu jeho hran, takže platí $Cena(T_S(U)) = |E_U|$. Vrcholy z množiny U se nazývají terminální vrcholy a vrcholy z množiny $V_U \setminus U$ se nazývají Steinerovy vrcholy.*

Hledání minimálního Steinerova stromu je ale stejně jako hledání nejkratší hamiltonovské cesty NP-těžký problém [32]. Existují ale aproximační algoritmy odhadující minimální Steinerovy stromy v polynomiálním čase. Pokud k-aproximační algoritmus nalezne Steinerův strom s cenou n, tak určitě platí, že nalezené řešení je maximálně k-krát horší než minimální Steinerův strom, neboli platí $n/k \leq min\{Cena(T_S)\}$. Jako přípustná heuristická funkce lze tedy využít cena takto aproximovaného minimálního Steinerova stromu vydělená číslem k. Tato heuristická funkce je určitě přípustná, protože platí $n/k \leq min\{Cena(T_S)\} \leq min\{Cena(H_P)\}$. Tímto způsobem je vypočítána hodnota heuristické funkce v algoritmu 4.1 na řádcích 24 a 38.

4.2 SMT-Hamiltonian-CBS (SMT-HCBS)

SMT-HCBS je další algoritmus optimalizující řešení MG-MAPF. Je to algoritmus založený na redukci problému na SAT a je odvozen z algoritmu SMT-CBS, který řeší MAPF. Stejně tedy kombinuje prohledávání stavového prostoru a redukci na problém SAT a také využívá línou konstrukci formulí. Od HCBS se liší především tím, že hledání permutace cílů a hledání cest, které cíle spojují, jsou řešeny najednou pomocí řešiče SAT.

Algoritmus pro instanci MG-MAPF $\Theta = (G, A, s_0, g)$ stejně jako SMT-CBS zavádí neúplnou formuli $\mathcal{H}(\Theta, t_M)$. Kvůli odlišné definici cíle, ale nemůže využívat grafy MDD. Jejich princip ale může být adaptován pro MG-MAPF zavedením hamiltonovských MDD, značených MDD_i^H, pro každého agenta $a_i \in A$. Hamiltonovský MDD je vytvořen stejným způsobem jako standardní MDD pomocí časové expanze grafu G, ale zavádí jiné podmínky pro obsažení vrcholů. Výsledný $MDD_i^H(G, t_M)$ obsahuje všechny možné hamiltonovské cesty začínající v agentově počátečním vrcholu a pokrývající všechny jeho cílové vrcholy, které jsou kratší než velikost makespanu t_M.

▶ **Definice 4.3.** *$MDD_i^H(G, t_M) = (V_i, E_i)$ je acyklický orientovaný graf vytvořený pro agenta a_i z grafu G expanzí do $t_M + 1$ časových kroků. Pro množinu jeho vrcholů platí:*

$$V_i = \{v^t \mid v \in V(G) \wedge t \in 0, 1, \ldots, t_M \wedge Dist(s_0(a_i), v) \leq t \wedge Cena(H_P(s_0(a_i), g(a_i) \cup \{v\})) \leq t_M\},$$

kde $Dist(u, v)$ značí vzdálenost mezi vrcholy u a v v grafu G a pro množinu jeho hran platí:

$$E_i = \{(u^t, v^{t+1}) \mid (\{u, v\} \in E(G) \vee u = v) \wedge u^t, v^{t+1} \in V_i\}.$$

Výpočet minimální hamiltonovské cesty je ale NP-těžký problém, takže konstrukce takových grafů není možná. Proto se zavádí relaxace MDD_i^H značená MDD_i^T, kde druhá podmínka pro obsažení vrcholu v v grafu je nahrazena existencí Steinerova stromu $T_S(s_0(a_i) \cup g(a_i) \cup v)$, jehož cena je maximálně rovna hodnotě makespanu. Výpočet Steinerova stromu je stále NP-těžký, ale jeho cena lze stejně jako v algoritmu HCBS nahradit cenou stromu vypočítaného k-aproximačním algoritmem. Existence hamiltonovské cesty jde potom zespodu odhadnout aproximovanou cenou Steinerova stromu následně vydělenou faktorem k. Vrchol $v \in V(G)$ je tedy obsažen v $MDD_i^T(G, t_M)$, pokud platí:

$$Cena(T_S(s_0(a_i) \cup g(a_i) \cup v)) \,/\, k \leq t_M,$$

kde cena Steinerova stromu T_S je vypočtena k-aproximačním algoritmem. Stejně jako pro graf MDD_i^H platí i pro MDD_i^T, že obsahuje všechny možné hamiltonovské cesty z agentova počátečního vrcholu pokrývající všechny cílové vrcholy, které jsou kratší než velikost makespanu. Od MDD_i^H se ale liší v tom, že může obsahovat vrcholy, přes které žádná taková cesta vést nemůže. [6]

4.2.1 Kódování do booleovské formule

Algoritmus SMT-HCBS využívá stejné kódování jaké bylo použito pro MAPF v algoritmech popsaných v minulé kapitole. Stejně jako SMT-CBS vynechává pravidla zakazující konflikty mezi agenty a postupně je přidává v průběhu algoritmu, dokud nenajde řešení nebo nevytvoří úplnou formuli. MG-MAPF má ale jinou definici cíle než MAPF, takže se musí nahradit i booleovské formule, které cíle kódují. Jsou nahrazeny následující formulí vytvořenou pro každý cílový vrchol $v \in g(a_i)$ obsažený v $MDD_i^T(G, t_M) = (V_i, E_i)$ a každého agenta $a_i \in A$:

$$\bigvee_{t \in 0,1,\ldots,t_M, v^t \in V_i} \mathcal{X}_v^t(a_i) \tag{4.1}$$

Kódování použité pro MAPF povoluje duplikace agentů v jednotlivých časových krocích. To pro problém MAPF nevadí, protože ohodnocená formule bude vždy obsahovat cestu z počátečního vrcholu do agentova cíle. Není ale možné použít stejný princip pro MG-MAPF, protože každý agent má více cílů a takové kódování nijak neověří, jestli byl každý cíl navštíven pravým agentem nebo jeho duplikátem. Musí se tedy navíc zavést nové omezení, které zajistí, že se agent neobjeví ve více vrcholech najednou. Toho jde například dosáhnout přidáním jednoho z následujících pravidel:

- Pokud se agent nachází v nějakém vrcholu, tak do něj musel vstoupit pomocí alespoň jedné hrany. Pro každou zavedenou proměnnou $\mathcal{X}_v^t(a_i)$, kde $t \neq 0$, je vytvořeno následující pravidlo:

$$\mathcal{X}_v^t(a_i) \implies \bigvee_{(u^{t-1}, v^t) \in E_i} \mathcal{E}_{u,v}^{t-1}(a_i) \tag{4.2}$$

- Agent se nesmí nacházet v jednom časovém kroku ve dvou vrcholech najednou. Pro každého agenta $a_i \in A$ a každý časový krok $t \in 0, 1, \ldots, t_M$ je vytvořeno následující pravidlo:

$$\bigwedge_{u^t, v^t \in V_i, u \neq v} \neg\mathcal{X}_u^t(a_i) \vee \neg\mathcal{X}_v^t(a_i) \tag{4.3}$$

4.2.2 Průběh algoritmu

SMT-HCBS funguje až na pár detailů úplně stejně jako algoritmus SMT-CBS. Stejně začíná s prázdnou množinou omezení zakazujících konflikty mezi agenty, kterou za běhu naplňuje. Jeho výhoda oproti HCBS při tvoření této množiny je, že se nevětví, takže dokáže rychleji konvergovat k platnému řešení. Algoritmus 4.2 obsahuje pseudokód popisující SMT-HCBS. Vnější cyklus (řádky 3–7) postupně zkouší zvětšující se hodnoty makespanu, dokud řešič SAT nenajde platné řešení. Algoritmus začíná makespan zkoušet od jeho minimální možné hodnoty, která je rovna délce nejdelší z nejkratších individuálních hamiltonovských cest (řádek 2).

Při hledání řešení pro daný makespan vytvoří SMT-HCBS nejprve neúplnou formuli $\mathcal{H}(\Theta, t_M)$ (řádek 9). Potom v cyklu tuto formuli líně rozšiřuje, dokud SAT řešič nenajde platné řešení (řádek 15) nebo vytvořená formule už není splnitelná (řádek 21). V takovém případě neexistuje řešení pro zadaný makespan a vnitřní cyklus končí. Pokud formule je splnitelná, tak musí algoritmus nejprve ověřit, že nalezené ohodnocení koresponduje s platným řešením MG-MAPF a neobsahuje žádné konflikty (řádek 14). Pokud řešení není platné a obsahuje nějaké konflikty, tak se do formule $\mathcal{H}(\Theta, t_M)$ musí přidat omezení, která nalezené konflikty zakáží (řádky 17–19). Výhoda tohoto líného přístupu k tvorbě formule je, že platné řešení může být nalezeno dříve, než je vytvořena úplná formule a řešič SAT dokáže vyřešit neúplnou formuli výrazně rychleji. [6]

Algoritmus 4.2: SMT-HCBS algoritmus řešící MG-MAPF

Input: $\Theta = (G, A, s_0, g)$
$konflikty \leftarrow \emptyset$
$t_M \leftarrow \max_{a_i \in A}\{\text{nejkratší hamiltonovská cesta z } s_0(a_i) \text{ pokrývající } g(a_i) \mid a_i \in A\}$
while *TRUE* **do**
  $(řešení, konflikty) \leftarrow$SMT-HCBS-Fixed$(\Theta, t_M, konflikty)$
  if $řešení \neq UNSAT$ **then**
    return *řešení*
  $t_M \leftarrow t_M + 1$
SMT-HCBS-Fixed $(\Theta, t_M, konflikty)$
  $\mathcal{H}(\Theta, t_M) \leftarrow$ zakódovat$(\Theta, t_M, konflikty)$
  while *TRUE* **do**
    $ohodnoecní \leftarrow$ vyřešit_SAT$(\mathcal{H}(\Theta, t_M))$
    if $ohodnocení \neq UNSAT$ **then**
      $řešení \leftarrow$ extrahovat_řešení$(ohodnocení)$
      $kolize \leftarrow$ validovat$(řešení)$
      if $kolize = \emptyset$ **then**
        return $(řešení, konflikty)$
      foreach $(a_i, a_j, v, t) \in kolize$ **do**
        $konflikty \leftarrow konflikty \cup \{(a_i, a_j, v, t)\}$
        $\mathcal{H}(\Theta, t_M) \leftarrow \mathcal{H}(\Theta, t_M) \cup \{\neg\mathcal{X}_v^t(a_i) \vee \neg\mathcal{X}_v^t(a_j)\}$
    else
      return *(UNSAT, konflikty)*

4.3 Porovnání algoritmů

Oba algoritmy byly otestovány na instancích MG-MAPF na 4-souvislých grafech ve tvaru mřížek. SMT-HCBS byl efektivnější na malých grafech, ale se zvyšujícím se počtem cílů na agenta byl jeho výkon dostižen a překonán algoritmem HCBS. Na větších grafech už byl HCBS mnohem efektivnější než SMT-HCBS, který se mu vůbec nedokázal rovnat. Výsledky takto dopadly z několika důvodů. Algoritmus SMT-HCBS musí při zvyšujícím se počtu cílů iterovat přes více nesplnitelných hodnot makespanu než SMT-CBS na instancích MAPF, protože v případech s mnoha cíli mezi sebou agenti více interagují a optimální hodnota makespanu je kvůli tomu vzdálenější hodnotě odhadnuté pomocí nejkratších individuálních hamiltonovských cest. Zároveň je při vyšším počtu cílových vrcholů ořezávání grafů MDD méně efektivní, protože v takových případech minimální Steinerovy stromy špatně odhadují opravdovou cenu hamiltonovských cest. Naopak HCBS dokáže lépe využít heuristické znalosti, protože odděluje volbu nejlepšího pořadí cílových vrcholů od hledání nejkratších cest mezi nimi. Toho nejde dosáhnout v SMT-HCBS, protože grafy MDD musí obsahovat všechny možné permutace cílových vrcholů. [6]

Kapitola 5

Modifikace SMT-HCBS

Jako součást této práce jsem implementoval algoritmus SMT-HCBS. Implementoval jsem ho s cílem dále ho modifikovat a zvýšit jeho efektivitu, protože předchozí testování, jehož výsledky jsem popsal v minulé kapitole, prokázalo, že SMT-HCBS je mnohem méně výkonný než HCBS. To platí především pro instance MG-MAPF na větších grafech nebo s větším počtem cílových vrcholů. Algoritmus jsem implementoval v jazyce C++ a pro řešení splnitelnosti vytvářených formulí jsem použil SAT řešič *Glucose* 4.2.1 [35]. Moje implementace minimalizuje makespan hledaného řešení a pracuje s definicí MG-MAPF, která zakazuje vrcholové a hranové konflikty.

SMT-HCBS jsem implementoval přesně podle algoritmu 4.2. Jeho pseudokód přesně nespecifikuje, jak algoritmus hledá nejkratší individuální hamiltonovské cesty a jak aproximuje cenu minimálního Steinerova stromu. V mé implementaci hledám nejkratší hamiltonovské cesty, které jsou používány k odhadnutí minimálního makespanu řešení, pomocí nižší úrovně algoritmu HCBS obsažené ve funkci *HCBS-Ordering* v algoritmu 4.1. Tato funkce prohledává prostor všech permutací agentových cílových vrcholů pomocí algoritmu A* a pokud má prázdnou množinu omezení, tak pro agenta najde nejkratší individuální hamiltonovskou cestu začínající v jeho počátečním vrcholu a pokrývající všechny jeho cíle.

K aproximaci ceny Steinerova stromu $T_S(U)$, která je využívána k tvorbě grafů MDD_i^T, používám 2-aproximační algoritmus uvedený v [50], jehož pseudokód je popsán algoritmem 5.1. Aproximační algoritmus nejprve vytvoří úplný ohodnocený graf G_U s množinou vrcholů U, jehož hrany jsou ohodnoceny podle vzdáleností mezi vrcholy v grafu G (řádky 1–3). Následně algoritmus najde minimální kostru grafu G_U (řádek 4), kterou v mé implementaci hledám pomocí Kruskalova algoritmu. Hrany minimální kostry grafu G_U reprezentují nejkratší cesty v grafu G, takže jsou jimi následně nahrazeny tak, aby nevznikly žádné cykly (řádky 6–14). Cyklus nevznikne, pokud cesta, která má nahradit hranu kostry, obsahuje nejvýše jeden vrchol, který už je obsažen v částečně zkonstruovaném Steinerově stromu (řádek 8). Jinak by ale cyklus vznikl, takže jsou do stromu přidány jen ty části cesty, které spojují terminální vrcholy s tvořeným grafem (řádky 11–14). Tímto způsobem se nepřidá žádná hrana, která by cyklus vytvořila, a výsledný graf zůstane stromem.

Tento algoritmus je 2-aproximační, takže cena nalezeného Steinerova stromu je maximálně dvakrát horší než jeho opravdová cena. Při konstrukci MDD_i^T je využita takto aproximovaná cena vydělená faktorem 2, aby nikdy nepřecenila cenu nejkratší hamiltonovské cesty přes stejné terminální vrcholy a SMT-HCBS stále hledal optimální řešení. Algoritmus aproximuje minimální Steinerův strom v čase $O(|V| \cdot |U|^2)$, kde V značí množinu vrcholů grafu G a U značí množinu terminálních vrcholů hledaného stromu [50]. Algoritmus je tedy polynomiální a doba jeho běhu výrazně neovlivní celkovou dobu běhu SMT-HCBS, jehož složitost je exponenciální.

Algoritmus 5.1: 2-aproximační algoritmus hledající minimální Steinerův strom

Input: $G = (V, E), U \subseteq V$
$E_U \leftarrow \{\{u, v\} \mid u, v \in U\}$
$w \leftarrow w\colon \{u, v\} \mapsto Dist(u, v)$
$G_U \leftarrow (U, E_U, w)$
$MST(U) \leftarrow$ minimální kostra grafu G_U
$T_S(U) \leftarrow \emptyset$
foreach $\{u, v\} \in E(MST(U))$ **do**
    $P \leftarrow$ nejkratší cesta z u do v v grafu G
    if P *obsahuje maximálně jeden vrchol z* $T_S(U)$ **then**
        $T_S(U) = T_S(U) \cup P$
    else
        $p_1 \leftarrow$ první vrchol z P obsažený v $T_S(U)$
        $p_2 \leftarrow$ poslední vrchol z P obsažený v $T_S(U)$
        $T_S(U) = T_S(U) \cup$ nejkratší cesta z u do p_1 v grafu G
        $T_S(U) = T_S(U) \cup$ nejkratší cesta z v do p_2 v grafu G
return $T_S(U)$

5.1 Původní kódování

Moje implementace pracuje s trochu jinou definicí MG-MAPF, než pro kterou jsem v předchozích kapitolách této práce uvedl kódování do booleovské formule. Pro přehlednost tedy uvedu konkrétní zakódování, které jsem použil. Neobsahuje žádné nové koncepty, pouze je upraveno pro detekci hranových konfliktů místo konfliktů následování a zvolil jsem jedno konkrétní pravidlo (ze dvou navržených v minulé kapitole) zakazující výskyt jednoho agenta ve více vrcholech najednou.

Předpokládejme instanci MG-MAPF $\Theta = (G, A, s_0, g)$ a fixní makespan t_M určený vnějším cyklem algoritmu SMT-HCBS. Kódování instance Θ do booleovské formule využívá princip MDD_i^T, takže je pro každého agenta $a_i \in A$ vytvořen graf $MDD_i^T(G, t_M) = (V_i, E_i)$. Na základě těchto grafů je vytvořen příslušný počet logických proměnných $\mathcal{X}_v^t(a_i)$ a $\mathcal{E}_{u,v}^t(a_i)$, které reprezentují výskyt agentů ve vrcholech a hranách. Proměnná $\mathcal{X}_v^t(a_i)$ je kladně ohodnocená, právě když se agent a_i nachází v časovém kroku t ve vrcholu v. Stejným způsobem funguje i ohodnocení proměnných $\mathcal{E}_{u,v}^t(a_i)$. SMT-HCBS využívá princip líné konstrukce formule, takže začíná s neúplnou formulí $\mathcal{H}(\Theta, t_M)$, která je vytvořena konjunkcí následujících pravidel. Nejprve uvádím pravidla kódující pohyb agentů v grafu:

- Pokud se agent nachází v nějakém vrcholu, tak ho musí opustit skrz právě jednu hranu. Pro každou zavedenou proměnnou $\mathcal{X}_u^t(a_i)$, kde $t \neq t_M$, jsou vytvořena následující dvě pravidla:

$$\mathcal{X}_u^t(a_i) \implies \bigvee_{(u^t, v^{t+1}) \in E_i} \mathcal{E}_{u,v}^t(a_i) \tag{5.1}$$

$$\bigwedge_{(u^t, v^{t+1}), (u^t, w^{t+1}) \in E_i} \neg\mathcal{E}_{u,v}^t(a_i) \vee \neg\mathcal{E}_{u,w}^t(a_i) \tag{5.2}$$

- Pokud agent použije nějakou hranu, tak do ní musí vstoupit a zase z ní vystoupit v jejích incidentních vrcholech. Pro každou zavedenou proměnnou $\mathcal{E}_{u,v}^t(a_i)$ je vytvořeno následující pravidlo:

$$\mathcal{E}_{u,v}^t(a_i) \implies \mathcal{X}_u^t(a_i) \wedge \mathcal{X}_v^{t+1}(a_i) \tag{5.3}$$

- Agent se v daný časový krok smí nacházet pouze v jednom vrcholu. Pro každého agenta $a_i \in A$ a každý časový krok $t \in 0, 1, \ldots, t_M$ je vytvořeno následující pravidlo:

$$\bigwedge_{u^t, v^t \in V_i, u \neq v} \neg\mathcal{X}_u^t(a_i) \vee \neg\mathcal{X}_v^t(a_i) \tag{5.4}$$

Dále jsou uvedena pravidla kódující konkrétní instanci MG-MAPF. Obsahují tedy počáteční konfigurace a cílové množiny každého agenta:

- Agent se musí v prvním časovém kroku nacházet ve svém počátečním vrcholu. Pro každého agenta $a_i \in A$ a jeho počáteční vrchol $v = s_0(a_i)$ je vytvořeno následující pravidlo:

$$\mathcal{X}_v^0(a_i) \tag{5.5}$$

- Agent musí na své cestě navštívit každý svůj cílový vrchol v jakémkoliv časovém kroku. Pro každého agenta $a_i \in A$ a každý jeho cílový vrchol $g \in g(a_i)$ je vytvořeno následující pravidlo:

$$\bigvee_{t \in 0,1,\ldots,t_M, g^t \in V_i} \mathcal{X}_g^t(a_i) \tag{5.6}$$

SAT řešič vyžaduje booleovskou formuli v CNF, takže pravidla nesmí obsahovat implikace. Každá implikace jde ale jednoduše převést do CNF, protože platí $(A \implies B) \iff (\neg A \vee B)$. Pokud B obsahuje pouze disjunkce, tak tímto způsobem vznikne jedna klauzule. Pokud B obsahuje konjunkce, tak jde formule $\neg A \vee B$ dále upravit pomocí pravidla distributivity a vznikne konjunkce několika klauzulí, která odpovídá CNF. Tímto způsobem jdou převést všechna výše uvedená pravidla a jejich konjunkcí vznikne booleovská formule $\mathcal{H}(\Theta, t_M)$, která je v CNF.

SMT-HCBS v průběhu hledání řešení přidává do neúplné formule $\mathcal{H}(\Theta, t_M)$ klauzule, které zakazují konflikty nalezené v době běhu algoritmu. Moje implementace zakazuje vrcholové a hranové konflikty pomocí těchto klauzulí:

- Pro každý vrcholový konflikt (a_i, a_j, v, t), který nastal mezí agenty a_i a a_j v časovém kroku t ve vrcholu v, je vytvořeno následující pravidlo:

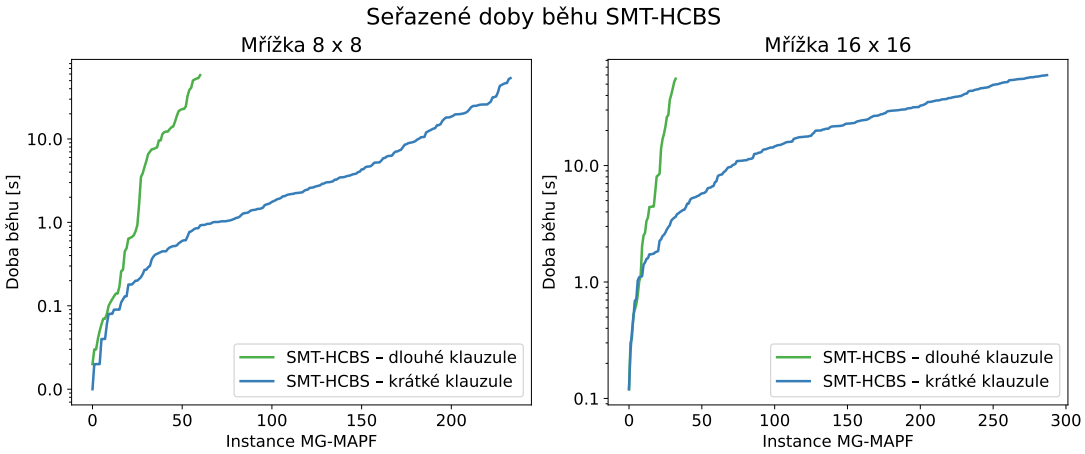
$$\neg\mathcal{X}_v^t(a_i) \vee \neg\mathcal{X}_v^t(a_j) \tag{5.7}$$

- Hranový konflikt (a_i, a_j, u, v, t) nastává mezí agenty a_i a a_j mezi časovými kroky t a $t+1$ v hraně $\{u, v\}$, skrze kterou se agent a_i přesouvá z vrcholu u do vrcholu v a agent a_j v opačném směru. Pro každý takový konflikt, je vytvořeno následující pravidlo:

$$\neg\mathcal{E}_{u,v}^t(a_i) \vee \neg\mathcal{E}_{v,u}^t(a_j) \tag{5.8}$$

Jediný rozdíl tohoto kódování od kódování popsaného v předchozích kapitolách této práce je ve formuli 5.5, která kóduje počáteční konfiguraci agentů. Formule se liší v tom, že obsahuje jedinou klauzuli s jediným literálem, která v prvním časovém kroku nastavuje ohodnocení právě jednoho vrcholu pro každého agenta. To je možné díky tomu, že kódování využívá grafy MDD, které na rozdíl od grafů TEG vždy obsahují právě jeden vrchol ve své první vrstvě.

V minulé kapitole této práce jsem uvedl dvě pravidla, která zajistí, že se agent v jednom časovém kroku nenachází ve dvou vrcholech zároveň. V mé implementaci jsem si vybral formuli 4.3, která pro každého agenta vytvoří $O(t_M \cdot n^2)$ klauzulí se dvěma literály, kde t_M značí makespan hledaného řešení a n značí počet vrcholů grafu G. Naopak formule 4.2 vytvoří pro každého agenta jen $O(t_M \cdot n)$ klauzulí, které ale obsahují až počet literálů odpovídající nejvyššímu stupni vrcholu v grafu G. Tato formule tedy tvoří méně klauzulí, ale každá může být výrazně delší než v prvním případě. Není tedy na první pohled jasné, který přístup je výhodnější. Já jsem použití obou formulí otestoval na malých grafech ve tvaru mřížek, jejichž maximální stupeň je čtyři. To je



Obrázek 5.1 Grafy porovnávající výkon SMT-HCBS na mřížkách při využití pravidla 4.2 nebo 4.3

docela nízký stupeň a klauzule vytvořené formulí 4.2 nejsou o mnoho delší něž klauzule vytvořené formulí 4.3.

Obrázek 5.1 obsahuje dva grafy, které použití těchto formulí porovnávají. Grafy obsahují vzestupně seřazené doby běhu SMT-HCBS na instancích MG-MAPF na dvou mřížkách bez překážek o rozměrech 8×8 a 16×16. Na menší mřížce jsem testoval instance s různými počty agentů a cílů v rozmezí 1–10. Na větší mřížce jsem testoval instance s počty agentů v rozmezí 1–25, kde každý agent měl dva cíle. Při testování na obou grafech měl algoritmus omezený časový limit pro nalezení řešení roven jedné minutě. Je vidět, že SMT-HCBS používající formuli 4.2, který je označený zelenou křivkou, je výrazně pomalejší na obou mřížkách a mnoho instancí ani nedokázal vyřešit. SMT-HCBS používající formuli 4.3 je mnohem efektivnější i na větší mřížce, zatímco druhá verze algoritmu je na ní v porovnání ještě pomalejší než na té menší. Tento rozdíl výkonu je nejspíše způsoben tím, že SAT řešič je efektivnější, když booleovská formule obsahuje co nejkratší klauzule, protože takové klauzule podporují jednotkovou propagaci.

5.2 Redukované kódování

Kódování SMT-HCBS se od kódování SMT-CBS liší pouze ve formuli 5.6 kódující cílové vrcholy, a navíc přidává formuli 5.4, která zakazuje agentům nacházet se ve více vrcholech zároveň. Zatímco se ale SMT-CBS na všech instancích testovaných v [49] minimálně vyrovnal algoritmu CBS, tak SMT-HCBS byl na většině instancí testovaných v [6] překonán algoritmem HCBS. Změna kódování, jejímž důsledkem je prodloužení výsledné booleovské formule, tedy negativně ovlivňuje výkonnost algoritmu v porovnání s algoritmy založenými na prohledávání stavového prostoru. Celkové době běhu SMT-HCBS dominuje doba běhu SAT řešiče, která závisí na velikosti zakódované formule. Efektivita algoritmu by tedy mohla být zvýšena, pokud by se redukovala velikost formulí vytvořených novými pravidly, a tak se zvýšila rychlost řešiče SAT.

Formule 5.6, která zachycuje rozmístění cílových vrcholů, nejde při zvoleném způsobu kódování pohybu agentů v grafu nijak změnit, protože není možné dopředu určit v jakých časových krocích budou cíle v optimálním řešení navštíveny. Pravidlo 5.4 zajišťuje, že ohodnocení logických proměnných vytvoří pro každého agenta cestu, která povede z jeho počátečního vrcholu přes všechny jeho cílové vrcholy. Zajistí to tak, že nepovolí agentům nacházet se ve více vrcholech zároveň v jednom časovém kroku. To ale není nutné pro to, aby bylo možné z nalezeného ohodnocení extrahovat platnou cestu pro každého agenta. Stačí zavést méně omezující pravidlo, které také dokáže zajistit platnost nalezeného řešení.

▶ **Tvrzení 5.1.** *Pokud se agent a_i smí nacházet ve více vrcholech najednou, jen když ani jeden z nich nepatří mezi jeho cílové vrcholy, tak platné ohodnocení booleovské formule $\mathcal{H}(\Theta, t_M)$, ze které je vynecháno pravidlo 5.4, vytvoří cestu, která začíná v agentově počátečním vrcholu a pokrývá všechny jeho cíle.*

Důkaz. Všechny cesty vytvořené ohodnocením formule $\mathcal{H}(\Theta, t_M)$ jsou díky pravidlu 5.1 souvislé a končí vždy až v časovém kroku t_M a pravidlo 5.5 zajišťuje, že existuje nějaká cesta, která začíná v agentově počátečním vrcholu $s_0(a_i)$ v nultém časovém kroku. Pravidlo 5.6 zajišťuje, že každý cílový vrchol $g \in g(a_i)$ je alespoň v jednom časovém kroku (označeném t_v) nalezeného řešení navštíven agentem a_i. Tvrzení 5.1 ale říká, že se agent a_i v časovém kroku t_v nemůže nacházet v žádném jiném vrcholu, protože g patří mezi jeho cílové vrcholy, takže souvislá cesta začínající v $s_0(a_i)$ musí vést v každý časový krok t_v skrz vrchol g. ◀

Formule 5.4 tedy vytváří nadbytečné klauzule, které mohou ztěžovat hledání platného ohodnocení formule $\mathcal{H}(\Theta, t_M)$. To stejné platí o formuli 5.2, která zakazuje duplikaci agentů ve výstupních hranách každého vrcholu. Zavádím tedy *redukované kódování* problému MG-MAPF, které dovoluje agentům nacházet se ve více vrcholech najednou, pokud ani jeden z nich nepatří do agentovy množiny cílových vrcholů. Redukované kódování obsahuje všechny formule původního kódování kromě formulí 5.2 a 5.4. Navíc je pro každého agenta $a_i \in A$, každý jeho cílový vrchol $g \in g(a_i)$ a každý časový krok $t \in 0, 1, \ldots, t_M$ takový, že platí $g^t \in V_i$, vytvořeno následující pravidlo:

$$\bigwedge_{v^t \in V_i, v \neq g} \neg \mathcal{X}_g^t(a_i) \vee \neg \mathcal{X}_v^t(a_i) \tag{5.9}$$

5.3 Líné redukované kódování

Redukovaná formule pořád obsahuje více klauzulí než booleovská formule používaná algoritmem SMT-CBS. Není už ale možné dále snížit počet klauzulí vytvořených pravidlem 5.9 tak, aby platné ohodnocení formule $\mathcal{H}(\Theta, t_M)$ vždy tvořilo řešení pokrývající všechny cílové vrcholy všech agentů. Po další redukci počtu těchto klauzulí by existence platného řešení pouze implikovala splnitelnost formule a neplatila by tedy požadovaná ekvivalence mezi existencí řešení a splnitelností formule. Formule $\mathcal{H}(\Theta, t_M)$ už ale je neúplná, protože neobsahuje klauzule zakazující konflikty mezi agenty, takže ekvivalence stejně neplatí a SMT-HCBS musí ověřovat platnost řešení vytvořeného z ohodnocení proměnných a líně konstruuje úplnou formuli.

Stejný princip je možné využít pro další redukci velikosti booleovské formule. Dále tedy zavádím *líné redukované kódování*, které využívá ještě více líný přístup ke konstrukci neúplné formule $\mathcal{H}(\Theta, t_M)$. Kromě pravidla 5.9 obsahuje stejná pravidla jako redukované kódování. V průběhu hledání řešení je formule líně dotvářena, dokud řešič SAT nenalezne platné řešení MG-MAPF nebo SMT-HCBS nepřidá všechny klauzule, které by jinak vytvořilo pravidlo 5.9. Výhoda tohoto přístupu je, že řešení může být nalezeno dříve, než jsou přidány všechny klauzule. SAT řešič dokáže navíc rychleji nalézt ohodnocení splňující kratší neúplnou formuli.

Algoritmus SMT-HCBS musí být pro využití tohoto kódování trochu upraven, protože řešení extrahované z ohodnocení formule $\mathcal{H}(\Theta, t_M)$ nemusí pokrývat všechny cílové vrcholy. Upravený algoritmus je z velké části stejný jako původní SMT-HCBS, jehož pseudokód je popsán v algoritmu 4.2. Po extrakci řešení (řádek 13) ale musí být zkontrolováno pokrytí všech cílových vrcholů. Pro každého agenta $a_i \in A$ a každý jeho cílový vrchol $g \in g(a_i)$ plyne z formule 5.6 to, že existuje alespoň jeden časový krok $t \in 0, 1, \ldots, t_M$ takový, že proměnná $\mathcal{X}_g^t(a_i)$ je ohodnocená kladně. Pokud cesta nalezená pro agenta a_i neobsahuje vrchol g, tak je do formule $\mathcal{H}(\Theta, t_M)$ pro každý takový časový krok t přidáno pravidlo 5.9.

Přidání tohoto pravidla způsobí, že se ohodnocení proměnných změní tak, že další nalezená cesta buď povede přes $\mathcal{X}_g^t(a_i)$, nebo tato proměnná už nebude ohodnocena kladně a cílový vrchol g bude navštíven v jiném časovém kroku. Přidávané klauzule takto postupně usměrňují nalezené

řešení, dokud nepokryje všechny cílové vrcholy. Implementoval jsem dvě varianty SMT-HCBS používající líné redukované kódování. První varianta nejprve hledá řešení pokrývající všechny cíle a až potom zkontroluje, jestli řešení obsahuje konflikty mezi agenty. Pokud tedy nalezené řešení všechny cíle nepokryje, tak je přeskočen cyklus přidávající konflikty do booleovské formule (řádky 17–19). Tím ale může být způsobeno to, že algoritmus bude dlouho hledat řešení, které sice navštíví všechny cíle, ale naplánované cesty obsahují konflikty. Druhá implementovaná varianta algoritmu průběžně do booleovské formule přidává i klauzule zakazující nalezené konflikty. Tímto rychleji usměrňuje nalezené řešení k platnosti, ale může přidat klauzule zakazující konflikty, které by při použití první varianty nebo původního kódování vůbec nevznikly. Může se tedy stát, že tato varianta rychleji najde platné řešení nebo naopak přidá zbytečně mnoho klauzulí a zbytečně sníží efektivitu SAT řešiče.

5.4 Porovnání délky vytvořených formulí

Uvedl jsem tři různé varianty kódování problému MG-MAPF do booleovské formule. Od sebe se liší ve způsobu využití pravidel zakazujících výskyt agentů ve více vrcholech najednou. Původní varianta kódování používá pravidlo 5.4, které vytvoří pro každého agenta $O(t_M \cdot n^2)$ klauzulí, kde t_M značí makespan hledaného řešení a n značí počet vrcholů grafu G, protože tvoří jednu klauzuli pro každou dvojici vrcholů v každém časovém kroku. Toto pravidlo má tedy výrazný vliv na délku výsledné formule a tím může mít vliv i na celkový výkon SMT-HCBS.

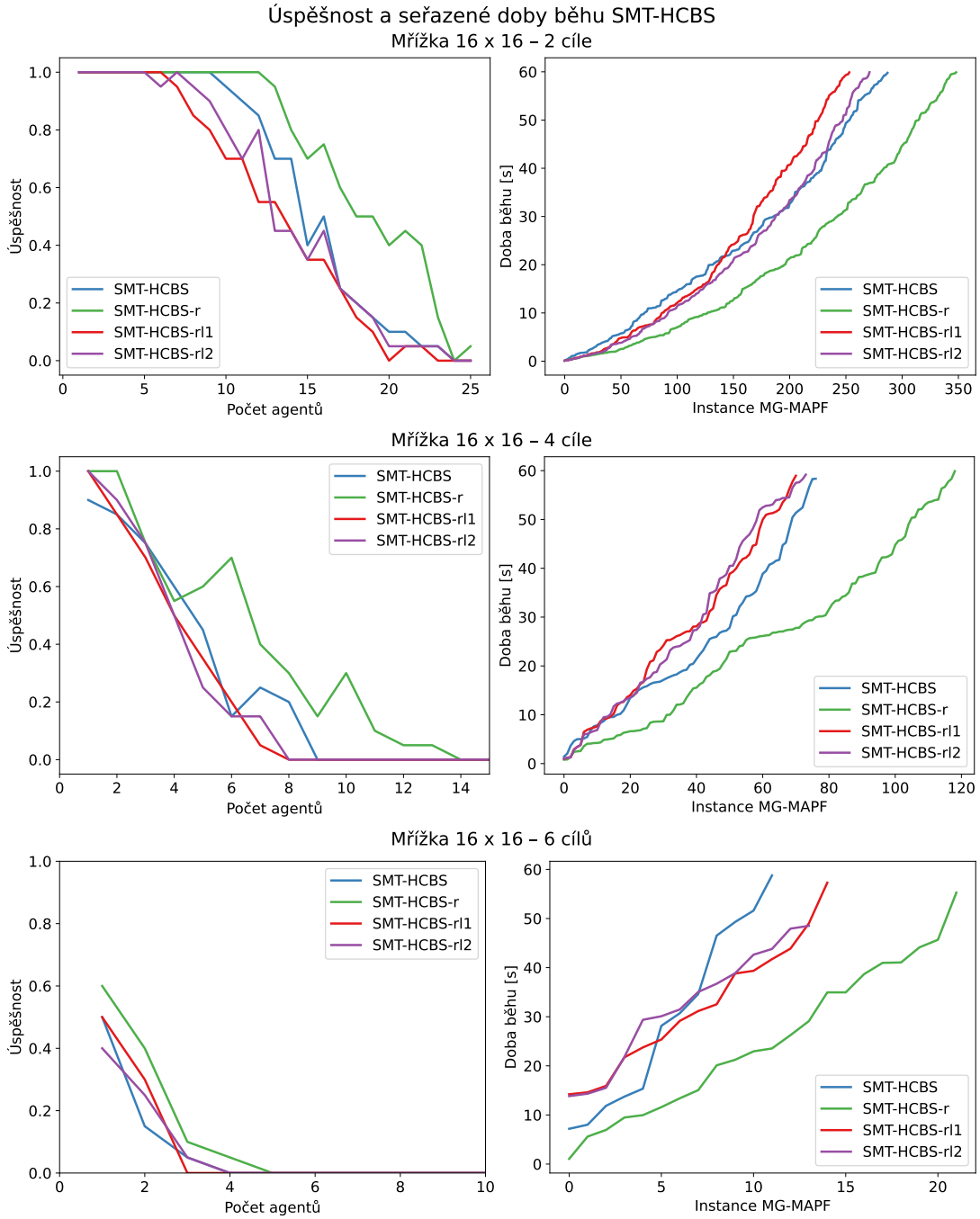
Redukované kódování vytvoří pomocí pravidla 5.9 pro každého agenta $O(t_M \cdot g \cdot n)$ klauzulí, kde g značí počet cílových vrcholů agenta, protože tvoří klauzule pouze pro dvojice vrcholů, z nichž je alespoň jeden cílový. Tyto klauzule obsahují vždy dva literály stejně jako klauzule vytvořené pravidlem 5.4, takže se vytvořená formule liší od formule vytvořené původním kódováním pouze v počtech klauzulí, které byly vygenerované zmíněnými dvěma pravidly. Čím méně klauzulí je kódováním vytvořeno, tím rychleji by měl SAT řešič najít ohodnocení splňující booleovskou formuli a tím efektivnější by měl být SMT-HCBS.

Formule vytvořená redukovanou variantou je z definice vždy kratší než formule vytvořená původním kódováním. Velikost rozdílu mezi jejich délkami záleží pouze na parametrech n a g, přičemž se rozdíl v délce projeví především na velkých grafech, protože počet klauzulí vygenerovaných pravidlem 5.4 roste kvadraticky s parametrem n. S rostoucím g se rozdíl délek zase zmenší, ale změna nebude tak výrazná, protože se rozdíl zmenší pouze lineárně. SMT-HCBS by tedy měl teoreticky být výkonnější při používání redukovaného kódování.

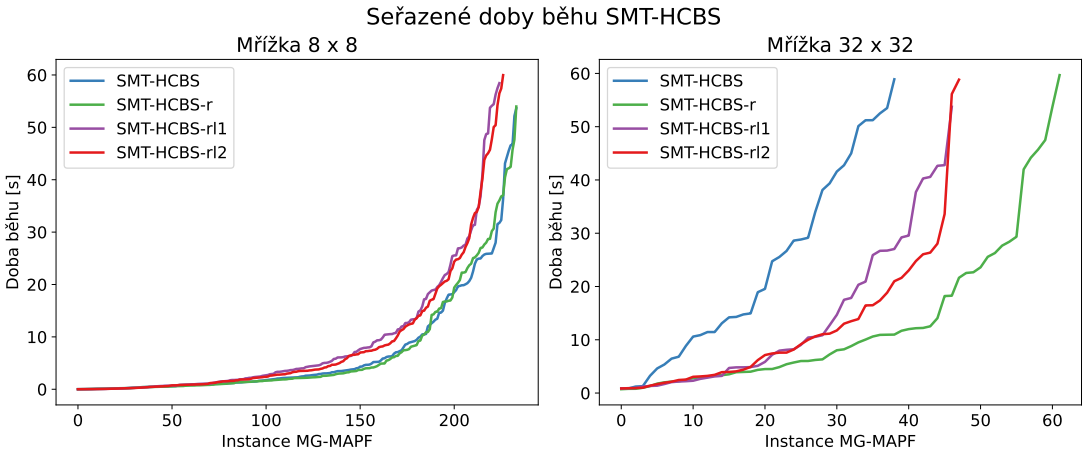
Líná varianta redukovaného kódování nejprve nevytvoří žádnou klauzuli, která by agentům zakazovala nacházet se ve více vrcholech zároveň. V průběhu algoritmu je ale booleovská formule postupně rozšiřována. V jedné iteraci cyklu na řádcích 10–21 v algoritmu 4.2 přidá SMT-HCBS využívající toto kódování pro každého agenta maximálně $O(t_M \cdot g \cdot n)$ klauzulí. To je stejný počet jako počet klauzulí vytvořených redukovaným kódováním pomocí pravidla 5.9. Není ale možné odhadnout opravdový počet přidaných klauzulí, protože záleží na konkrétním nalezeném ohodnocení booleovské formule a na tom, kolik cílových vrcholů bylo tímto řešením pokryto. Toto kódování by teoreticky mohlo být lepší něž předchozí dvě, především na velkých grafech, ale zároveň se může stát, že SMT-HCBS používající toto kódování bude iterovat přes mnoho neplatných řešení a jeho efektivita se naopak zhorší.

5.5 Experimenty

Uvedená kódování jsem otestoval na instancích MG-MAPF založených na instancích testovaných v [6], ve kterých byli agenti umístěni ve 4-souvislých grafech ve tvaru mřížek. Jejich konkrétní rozložení bylo vygenerované na základě instancí MAPF z [8]. Každý agent má ale v problému MAPF přiřazen jediný cílový vrchol, takže aby vytvořené instance zohledňovaly celou problematiku MG-MAPF, byly agentům přiřazeny další cíle pomocí náhodného výběru vrcholů z cílů



Obrázek 5.2 Grafy porovnávající výkon SMT-HCBS na mřížce o rozměrech 16×16 s různými variantami kódování booleovské formule



Obrázek 5.3 Grafy porovnávající výkon SMT-HCBS s různými variantami kódování booleovské formule na dvou mřížkách

ostatních agentů. Díky tomu má každý agent své cílové vrcholy rovnoměrně rozmístěné po celém grafu. Instance jsem generoval pomocí konvertoru z [51], který takto transformoval instance MAPF.

Na vytvořených instancích jsem otestoval algoritmus SMT-HCBS používající jeho původní kódování (SMT-HCBS), redukované kódování (SMT-HCBS-r) a dvě varianty líného redukovaného kódování. První varianta algoritmu používající líné kódování přidává klauzule zakazující konflikty mezi agenty až po nalezení řešení pokrývajícího všechny cíle (SMT-HCBS-rl1) a druhá varianta tyto klauzule přidává průběžně (SMT-HCBS-rl2). Na všech testovaných instancích jsem spouštěl algoritmus s časovým limitem jedné minuty pro nalezení optimálního řešení.

Mřížka 16 × 16 Nejprve jsem otestoval uvedené varianty kódování na prázdných mřížkách o rozměrech 16 × 16. Rozdělil jsem vygenerované instance MG-MAPF na těchto mřížkách podle velikosti cílových množin agentů na tři skupiny (2, 4 a 6 cílových vrcholů). S takto fixním počtem cílů jsem testoval zadání s různými počty agentů a měřil úspěšnost SMT-HCBS a dobu jeho běhu. Výsledky jsou zobrazené na obrázku 5.2. Grafy v levém sloupci ilustrují úspěšnost variant algoritmu, kterou jsem měřil pro fixní počet agentů v testované instanci. Úspěšnost byla změřena z dvaceti odlišných zadání pro všechny počty agentů v rozmezí 1–25 jako poměr úspěšně nalezených řešení v zadaném časovém limitu vůči počtu testovaných instancí. Grafy v pravém sloupci obsahují pro každou variantu SMT-HCBS vzestupně seřazené doby jeho běhu na úspěšně vyřešených instancích MG-MAPF.

Výsledky ukazují, že redukované kódování bylo opravdu ve všech případech lepší než původní kódování, ale stejně, jako úspěšnost ostatních variant, i jeho úspěšnost poměrně rychle klesala se zvyšujícím se počtem cílových vrcholů. Se šesti cílovými vrcholy už dokázaly všechny varianty kódování vyřešit jen ta nejlehčí zadání. Ani jedna varianta SMT-HCBS s líným kódováním původní kódování nepřekonala a na instancích se čtyřmi cíli byly naopak obě varianty trochu horší.

Mřížka 8 × 8 Dále jsem otestoval všechny varianty kódování na mřížkách jiných velikostí. Na prázdné mřížce o rozměrech 8 × 8 jsem otestoval instance s počty agentů i cílových vrcholů v rozmezí 1–10. Na levém grafu na obrázku 5.3 jsou zobrazeny seřazené doby běhu algoritmu SMT-HCBS používajícího různá kódování. Všechny varianty algoritmu vyřešily skoro všechny zadané instance a jejich doby běhu byly velmi podobné. Redukované kódování tedy na takto malé mřížce nepřineslo žádné zlepšení. Líné varianty byly dokonce trochu horší, což bylo nejspíš způsobeno zvýšenou režií spojenou s průběžnou konstrukcí formule.

Mřížka 32 × 32 Poslední graf, na kterém jsem testoval efektivitu uvedených kódování, byla mřížka o rozměrech 32 × 32, ze které bylo vynecháno 20 % náhodně vybraných vrcholů. Testoval jsem na ní instance s počty agentů v rozmezí 1–8 a počty cílových vrcholů každého agenta v rozmezí 1–4. Obrázek 5.3 zobrazuje výsledky i tohoto měření. Na této mřížce dokázal SMT-HCBS vyřešit se všemi variantami kódování méně instancí než na ostatních mřížkách, protože s velikostí grafu roste obtížnost problému. Rozdíl mezi efektivitou původního a redukovaného kódování byl ale výraznější než na menších mřížkách. Algoritmu používajícímu původní kódování navíc při testování na nejtěžších instancích docházela paměť, které měl k dispozici 8 GB. Na této mřížce byly líné varianty poprvé lepší než původní kódování, ale stále se nedokázaly vyrovnat redukované variantě.

Redukované kódování dokázalo zvýšit efektivitu SMT-HCBS podle očekávání především na větších grafech. Největší zlepšení přineslo na největší mřížce, zatímco na nejmenší bylo stejně efektivní jako původní kódování. Tento výsledek je způsoben tím, že původní kódování přidává navíc kvadraticky mnoho klauzulí vůči počtu vrcholů grafu. Obě varianty SMT-HCBS používající líné redukované kódování byly přibližně stejně výkonné, ale efektivitu původního kódování se jim dokázalo překonat pouze na největší testované mřížce. Na ostatních grafech byly dokonce horší než původní kódování nejspíše kvůli své zvýšené režii, která plyne z líné konstrukce booleovské formule a opakovaného spouštění SAT řešiče. K efektivitě redukovaného kódování se ani nedokázaly přiblížit.

Závěr

V teoretické části této práce jsem definoval problémy MAPF a MG-MAPF, jejichž cílem je naplánovat pro každého agenta cestu podle svého zadání. Nalezené řešení je ale platné pouze pokud cesty neobsahují žádné konflikty mezi agenty. Kvalita řešení je určena podle hodnoty účelové funkce, jejíž nejpoužívanější zástupci jsou makespan a sum of costs. V dalších kapitolách jsem analyzoval existující algoritmy hledající řešení těchto problémů. Algoritmy mohou hledat suboptimální řešení, které je možné najít v polynomiálním čase, nebo optimální řešení, jehož hledání je NP-těžký problém.

Algoritmy optimalizující MAPF a MG-MAPF jsou založené buď na prohledávání stavového prostoru nebo na redukci na jiný problém. V mé práci jsem se především věnoval algoritmům redukujícím MG-MAPF na SAT. Tyto algoritmy používají pro hledání řešení velmi výkonné SAT řešiče, které ale nemohou využívat žádné heuristické znalosti o konkrétním zadání. Tyto znalosti mohou být využity algoritmy prohledávajícími stavový prostor jako algoritmus HCBS. Mezi nejpokročilejší algoritmy založené na redukci na výrokovou splnitelnost patří algoritmus SMT-HCBS.

V praktické části práce jsem tento algoritmus implementoval. Jeho efektivita byla v předchozím testování výrazně nižší než efektivita HCBS, především na instancích s většími grafy a s vyšším počtem cílů. SMT-HCBS potřebuje kódování zadání MG-MAPF v booleovské formuli, s jejíž délkou klesá výkonnost SAT řešiče. Pro efektivitu algoritmu je tedy nejvýhodnější vytvořit co nejkratší formuli, která navíc obsahuje krátké klauzule. Uvedl jsem tedy dvě nové varianty kódování problému MG-MAPF, které tvoří kratší booleovskou formuli. Takto modifikovaný SMT-HCBS jsem otestoval na různých grafech ve tvaru mřížek. Algoritmus využívající redukovanou variantu kódování byl na všech testovaných grafech efektivnější než původní implementace, což se ukázalo především na největších testovaných mřížkách. SMT-HCBS využívající línou variantu redukovaného kódování ale efektivnější většinou nebyl a na některých grafech byl dokonce trochu pomalejší než původní implementace. To bylo nejspíš způsobenou zvýšeným množstvím režie potřebné pro využití tohoto kódování.

Modifikovaný algoritmus ale stále generuje dlouhou formuli. Další možná úprava, která by mohla vést ke zvýšení efektivity SMT-HCBS, by mohla být redukce velikosti vytvořených MDD, ze kterých vychází kódování booleovské formule. Toho by šlo dosáhnout lepším odhadem délky okružních cest, než je odhad 2-aproximačním algoritmem hledajícím minimální Steinerův strom. Je možné použít nějaký algoritmus, který dokáže Steinerův strom odhadnout s menším aproximačním faktorem, nebo použít algoritmus, který přímo odhaduje délku nejkratší hamiltonovské cesty. Také by mohlo být možné konstruovat booleovskou formuli ještě líněji nějakým jiným způsobem, než jsem zkoušel já. Například vynecháním nějakých klauzulí kódujících pohyb agentů nebo postupnou konstrukcí MDD.

Bibliografie

1. WURMAN, Peter R.; D'ANDREA, Raffaello; MOUNTZ, Mick. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine*. 2008, roč. 29, č. 1. Dostupné z DOI: 10.1609/aimag.v29i1.2082.
2. MORRIS, Robert; PASAREANU, Corina S.; LUCKOW, Kasper Søe; MALIK, Waqar A.; MA, Hang; KUMAR, T. K. Satish; KOENIG, Sven. Planning, Scheduling and Monitoring for Airport Surface Operations. In: *AAAI Workshop: Planning for Hybrid Systems*. 2016. Dostupné také z: https://www.researchgate.net/publication/296706370_Planning_Scheduling_and_Monitoring_for_Airport_Surface_Operations.
3. VELOSO, Manuela M.; BISWAS, Joydeep; COLTIN, Brian; ROSENTHAL, Stephanie. CoBots: Robust Symbiotic Autonomous Mobile Service Robots. In: *Proceedings of the 24th International Conference on Artificial Intelligence*. AAAI Press, 2015, s. 4423–4429. Dostupné také z: https://dl.acm.org/doi/abs/10.5555/2832747.2832901.
4. MA, Hang; YANG, Jingxing; COHEN, Liron; KUMAR, T. K. Satish; KOENIG, Sven. Feasibility Study: Moving Non-Homogeneous Teams in Congested Video Game Environments. In: *Artificial Intelligence and Interactive Digital Entertainment Conference*. 2017, sv. 13, s. 270–272. Č. 1. Dostupné z DOI: 10.1609/aiide.v13i1.12919.
5. YU, Jingjin; LAVALLE, S.M. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. *Proceedings of the 27th AAAI Conference on Artificial Intelligence*. 2013, roč. 27, s. 1443–1449. Dostupné z DOI: 10.1609/aaai.v27i1.8541.
6. SURYNEK, Pavel. Multi-Goal Multi-Agent Path Finding via Decoupled and Integrated Goal Vertex Ordering. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2021, roč. 35, č. 14, s. 12409–12417. Dostupné z DOI: 10.1609/aaai.v35i14.17472.
7. SURYNEK, Pavel; FELNER, Ariel; STERN, Roni; BOYARSKI, Eli. Efficient SAT approach to multi-agent path finding under the sum of costs objective. In: *Proceedings of the twenty-second european conference on artificial intelligence*. IOS Press, 2016, s. 810–818. Dostupné také z: https://dl.acm.org/doi/abs/10.3233/978-1-61499-672-9-810.
8. STERN, Roni; STURTEVANT, Nathan R.; FELNER, Ariel; KOENIG, Sven; MA, Hang; WALKER, Thayne T.; LI, Jiaoyang; ATZMON, Dor; COHEN, Liron; KUMAR, T. K. Satish; BOYARSKI, Eli; BARTÁK, Roman. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. *Symposium on Combinatorial Search (SoCS)*. 2019, s. 151–158. Dostupné z DOI: 10.1609/socs.v10i1.18510.
9. FELNER, Ariel; STERN, Roni; KRAUS, Sarit; BEN-YAIR, Asaph; NETANYAHU, Nathan S. PHA*: Finding the Shortest Path with A* in An Unknown Physical Environment. *J. Artif. Intell. Res.* 2004, roč. 21, s. 631–670. Dostupné z DOI: 10.1613/jair.1373.

10. LAVALLE, S.M.; HUTCHINSON, S.A. Optimal motion planning for multiple robots having independent goals. *IEEE Transactions on Robotics and Automation*. 1998, roč. 14, č. 6, s. 912–925. Dostupné z DOI: 10.1109/70.736775.
11. SHARON, Guni; STERN, Roni; FELNER, Ariel; STURTEVANT, Nathan R. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*. 2015, roč. 219, s. 40–66. Dostupné z DOI: 10.1016/j.artint.2014.11.006.
12. YU, Jingjin; LAVALLE, Steven M. Multi-agent Path Planning and Network Flow. In: *Springer Tracts in Advanced Robotics*. Springer, 2013, s. 157–173. Dostupné z DOI: 10.1007/978-3-642-36279-8_10.
13. YU, Jingjin; LAVALLE, Steven M. Planning optimal paths for multiple robots on graphs. In: *IEEE International Conference on Robotics and Automation*. 2013, s. 3612–3617. Dostupné z DOI: 10.1109/ICRA.2013.6631084.
14. BARTÁK, Roman; ŠVANCARA, Jiří; VLK, Marek. A Scheduling-Based Approach to Multi-Agent Path Finding with Weighted and Capacitated Arcs. In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. Stockholm, Sweden: International Foundation for Autonomous Agents a Multiagent Systems, 2018, s. 748–756. AAMAS '18. Dostupné také z: https://dl.acm.org/doi/10.5555/3237383.3237494.
15. WALKER, Thayne T.; STURTEVANT, Nathan R.; FELNER, Ariel. Extended Increasing Cost Tree Search for Non-Unit Cost Domains. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 2018, s. 534–540. Dostupné z DOI: 10.24963/ijcai.2018/74.
16. ATZMON, Dor; STERN, Roni; FELNER, Ariel; WAGNER, Glenn; BARTÁK, Roman; ZHOU, Neng-Fa. Robust multi-agent path finding. In: *Proceedings of the International Symposium on Combinatorial Search*. 2018, sv. 9, s. 2–9. Č. 1. Dostupné z DOI: 10.1609/socs.v9i1.18445.
17. WAGNER, Glenn; CHOSET, Howie. Path Planning for Multiple Agents under Uncertainty. *Proceedings of the International Conference on Automated Planning and Scheduling*. 2017, roč. 27, č. 1, s. 577–585. Dostupné z DOI: 10.1609/icaps.v27i1.13866.
18. BAREL, Ariel; MANOR, Rotem; BRUCKSTEIN, Alfred M. COME TOGETHER: Multi-Agent Geometric Consensus (Gathering, Rendezvous, Clustering, Aggregation). *CoRR*. 2019, roč. abs/1902.01455. Dostupné z DOI: 10.48550/arXiv.1902.01455.
19. LI, Jiaoyang; SURYNEK, Pavel; FELNER, Ariel; MA, Hang; KUMAR, T.; KOENIG, Sven. Multi-Agent Path Finding for Large Agents. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2019, roč. 33, s. 7627–7634. Dostupné z DOI: 10.1609/aaai.v33i01.33017627.
20. HÖNIG, Wolfgang; PREISS, James A.; KUMAR, T. K. Satish; SUKHATME, Gaurav S.; AYANIAN, Nora. Trajectory Planning for Quadrotor Swarms. *IEEE Transactions on Robotics*. 2018, roč. 34, č. 4, s. 856–869. Dostupné z DOI: 10.1109/TRO.2018.2853613.
21. HÖNIG, Wolfgang; KUMAR, T.; COHEN, Liron; MA, Hang; XU, Hong; AYANIAN, Nora; KOENIG, Sven. Summary: Multi-Agent Path Finding with Kinematic Constraints. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. Melbourne, Australia: AAAI Press, 2017, s. 4869–4873. Dostupné z DOI: 10.24963/ijcai.2017/684.
22. KLODER, S.; HUTCHINSON, S. Path planning for permutation-invariant multirobot formations. *IEEE Transactions on Robotics*. 2006, roč. 22, č. 4, s. 650–665. Dostupné z DOI: 10.1109/TRO.2006.878952.
23. MA, Hang; KOENIG, Sven. Optimal Target Assignment and Path Finding for Teams of Agents. *CoRR*. 2016, roč. abs/1612.05693. Dostupné z DOI: 10.48550/arXiv.1612.05693.

24. SOLOVEY, Kiril; HALPERIN, Dan. K-Color Multi-Robot Motion Planning. *The International Journal of Robotics Research*. 2014, roč. 33, č. 1, s. 82–97. Dostupné z DOI: 10.1177/0278364913506268.

25. ŠVANCARA, Jiří; VLK, Marek; STERN, Roni; ATZMON, Dor; BARTÁK, Roman. Online Multi-Agent Pathfinding. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2019, roč. 33, s. 7732–7739. Dostupné z DOI: 10.1609/aaai.v33i01.33017732.

26. MA, Hang; LI, Jiaoyang; KUMAR, T. K. Satish; KOENIG, Sven. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents a Multiagent Systems, 2017, sv. abs/1705.10868, s. 837–845. Dostupné také z: http://arxiv.org/abs/1705.10868.

27. HONG, Ted; LI, Yanjing; PARK, Sung-Boem; MUI, Diana; LIN, David; KALEQ, Ziyad Abdel; HAKIM, Nagib; NAEIMI, Helia; GARDNER, Donald S.; MITRA, Subhasish. QED: Quick Error Detection tests for effective post-silicon validation. In: *2010 IEEE International Test Conference*. 2010, s. 1–10. Dostupné z DOI: 10.1109/TEST.2010.5699215.

28. GUPTA, Aarti; GANAI, Malay K.; WANG, Chao. SAT-Based Verification Methods and Applications in Hardware Verification. In: *Formal Methods for Hardware Verification*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, s. 108–143. SFM'06. Dostupné z DOI: 10.1007/11757283_5.

29. TSEITIN, G. S. On the Complexity of Derivation in Propositional Calculus. In: *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*. Ed. SIEKMANN, Jörg H.; WRIGHTSON, Graham. Berlin, Heidelberg: Springer Berlin Heidelberg, 1983, s. 466–483. Dostupné z DOI: 10.1007/978-3-642-81955-1_28.

30. BIERE, A.; HEULE, M.; MAAREN, H. van; WALSH, T. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. NLD: IOS Press, 2009. ISBN 1586039296.

31. COOK, Stephen A. The Complexity of Theorem-Proving Procedures. In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. New York, NY, USA: Association for Computing Machinery, 1971, s. 151–158. STOC '71. Dostupné z DOI: 10.1145/800157.805047.

32. KARP, Richard M. Reducibility among Combinatorial Problems. In: *Complexity of Computer Computations*. Ed. MILLER, Raymond E.; THATCHER, James W.; BOHLINGER, Jean D. Boston, MA: Springer US, 1972, s. 85–103. ISBN 978-1-4684-2001-2. Dostupné z DOI: 10.1007/978-1-4684-2001-2_9.

33. DAVIS, Martin; LOGEMANN, George; LOVELAND, Donald. A Machine Program for Theorem-Proving. *Commun. ACM*. 1962, roč. 5, č. 7, s. 394–397. Dostupné z DOI: 10.1145/368273.368557.

34. EÉN, Niklas; SÖRENSSON, Niklas. An Extensible SAT-solver. In: GIUNCHIGLIA, Enrico; TACCHELLA, Armando (ed.). *Theory and Applications of Satisfiability Testing*. Springer Berlin Heidelberg, 2004, s. 502–518. Dostupné z DOI: 10.1007/978-3-540-24605-3_37.

35. AUDEMARD, Gilles; SIMON, Laurent. On the Glucose SAT Solver. *International Journal on Artificial Intelligence Tools*. 2018, roč. 27, č. 01, s. 1–25. Dostupné z DOI: 10.1142/S0218213018400018.

36. SILVER, David. Cooperative Pathfinding. In: *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Marina del Rey, California: AAAI Press, 2005, s. 117–122. Dostupné z DOI: 10.1609/aiide.v1i1.18726.

37. LUNA, Ryan; BEKRIS, Kostas E. Efficient and complete centralized multi-robot path planning. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, s. 3268–3275. Dostupné z DOI: 10.1109/IROS.2011.6095085.

38. KHORSHID, Mokhtar; HOLTE, Robert; STURTEVANT, Nathan. A Polynomial-Time Algorithm for Non-Optimal Multi-Agent Pathfinding. In: *Proceedings of the 4th Annual Symposium on Combinatorial Search, SoCS 2011*. 2011, sv. 2, s. 76–83. Č. 1. Dostupné z DOI: 10.1609/socs.v2i1.18205.

39. STANDLEY, Trevor. Finding optimal solutions to cooperative pathfinding problems. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI Press, 2010, sv. 24, s. 173–178. Č. 1. Dostupné z DOI: 10.1609/aaai.v24i1.7564.

40. WAGNER, Glenn; CHOSET, Howie. M*: A complete multirobot path planning algorithm with performance bounds. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, s. 3260–3267. Dostupné z DOI: 10.1109/IROS.2011.6095022.

41. GOLDENBERG, Meir; FELNER, Ariel; STERN, Roni; SHARON, Guni; STURTEVANT, Nathan; HOLTE, Robert C; SCHAEFFER, Jonathan. Enhanced partial expansion A*. *Journal of Artificial Intelligence Research*. 2014, roč. 50, s. 141–187. Dostupné z DOI: 10.1613/jair.4171.

42. SHARON, Guni; STERN, Roni; GOLDENBERG, Meir; FELNER, Ariel. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial intelligence*. 2013, roč. 195, s. 470–495. Dostupné z DOI: 10.1016/j.artint.2012.11.006.

43. SURYNEK, Pavel. Time-expanded graph-based propositional encodings for makespan-optimal solving of cooperative path finding problems. *Annals of Mathematics and Artificial Intelligence*. 2017, roč. 81, s. 329–375. Dostupné z DOI: 10.1007/s10472-017-9560-z.

44. SURYNEK, Pavel. Simple direct propositional encoding of cooperative path finding simplified yet more. In: *Nature-Inspired Computation and Machine Learning*. Springer, 2014, s. 410–425. Dostupné z DOI: 10.1007/978-3-319-13650-9_36.

45. SURYNEK, Pavel. Towards optimal cooperative path planning in hard setups through satisfiability solving. In: *PRICAI 2012: Trends in Artificial Intelligence*. Springer, 2012, s. 564–576. Dostupné z DOI: 10.1007/978-3-642-32695-0_50.

46. YU, Jingjin; LAVALLE, Steven M. Planning optimal paths for multiple robots on graphs. In: *International Conference on Robotics and Automation*. IEEE, 2013, s. 3612–3617. Dostupné z DOI: 10.1109/ICRA.2013.6631084.

47. ERDEM, Esra; KISA, Doga G.; OZTOK, Umut; SCHÜLLER, Peter. A General Formal Framework for Pathfinding Problems with Multiple Agents. In: *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*. Bellevue, Washington: AAAI Press, 2013, sv. 27, s. 290–296. AAAI'13, č. 1. Dostupné z DOI: 10.1609/aaai.v27i1.8592.

48. WYNN, Ed. A comparison of encodings for cardinality constraints in a SAT solver. *CoRR*. 2018, roč. abs/1810.12975. Dostupné z DOI: 10.48550/arXiv.1810.12975.

49. SURYNEK, Pavel. Unifying Search-based and Compilation-based Approaches to Multi-agent Path Finding through Satisfiability Modulo Theories. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, 2019, s. 1177–1183. Dostupné z DOI: 10.24963/ijcai.2019/164.

50. WU, Bang Ye; CHAO, Kun-Mao. *Spanning trees and optimization problems*. Abingdon: Chapman a Hall, 2004. ISBN 9780203497289.

51. SURYNEK, Pavel. *boOX* [soft.]. GitHub, 2018 [cit. 2023-04-30]. Dostupné z: https://github.com/surynek/boOX.

Obsah přiloženého archivu

- `assignments` ..... adresář obsahující soubory se zadáními problému MG-MAPF
- `build.sh` ..... bashový skript pro sestavení spustitelného programu
- `CMakeLists.txt` ..... soubor specifikující strukturu projektu pro nástroj *CMake*
- `license.txt` ..... soubor obsahující licenční ujednání
- `README.md` ..... stručný popis archivu a spuštění programu
- `src` ..... adresář se zdrojovými kódy implementace
- `thesis`
  - `src` ..... zdrojový kód práce ve formátu LaTeX
  - `thesis.pdf` ..... text práce ve formátu PDF