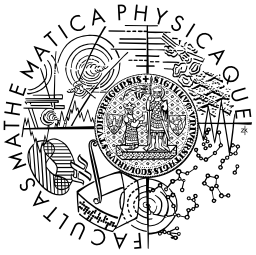


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

DIPLOMOVÁ PRÁCE

Jiří Švancara

Multi-agentní hledání cest v orientovaných prostředích

Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce:	RNDr. Pavel Surynek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Teoretická informatika

Praha 2016

Prohlašuji, že jsem tuto diplomovou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

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

V ..... dne ..... Podpis autora

Název práce: Multi-agentní hledání cest v orientovaných prostředích

Autor: Jiří Švancara

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: RNDr. Pavel Surynek, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: V této práci se zaměříme na optimální multi-agentní plánování cest, které je NP-úplným problémem. K řešení této úlohy budeme využívat centralizovaný prohledávací algoritmus A*, pro který navrhneme novou heuristiku. Ke konstrukci námi navržené heuristiky zkoumáme řešení multi-agentního plánování pomocí převodu na multi-komoditní tok, který je také NP-úplným problémem. Naše heuristika spočívá v relaxaci multi-komoditního toku na jedno-komoditní tok, který lze řešit v polynomiálním čase. Ukážeme, že takto vybudovaná heuristika je přípustná a konzistentní. Dále také ukážeme typy zadání, pro které je naše heuristika úspěšnější než jiné heuristiky.

Klíčová slova: Multi-agentní plánování cest, Algoritmus A*, Heuristika, Toky v sítích

Title: Multi-agent Path Planning in Unidirectional Environments

Author: Jiří Švancara

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. Pavel Surynek, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: In this paper we focus on the optimal multi-agent path planning, which is an NP-complete problem. To solve this task, we will use centralized A* search algorithm for which we propose a new heuristic. To construct our proposed heuristic, we examine the solution of multi-agent path planning via its reduction to multi-commodity flow, which is also an NP-complete problem. Our heuristic is based on relaxation of multi-commodity flow to single-commodity flow that can be solved in polynomial time. We show that this new heuristics is admissible and consistent. We also show types of problems for which our heuristic is more successful than other heuristics.

Keywords: Multi-Agent Path Planning, A* Algorithm, Heuristic, Flow Network

Rád bych poděkoval svému vedoucímu práce RNDr. Pavlu Surynkovi, Ph.D., že mi věnoval svůj čas a cenné rady, bez nichž bych tuto práci nemohl dokončit. Také chci poděkovat svým kolegům a přátelům, kteří mě podporovali.

Obsah

1. Úvod

Multi-agentní plánování je důležitým výpočetním problémem, který nachází uplatnění v oblasti robotiky, strategických hrách, simulacích, efektivním organizování skladišť a další. Není proto divu, že v posledních letech tento problém získal pozornost odvětví jako je umělá inteligence a robotika [9] [22] [21]. Obecným úkolem multi-agentního plánování je navigace dvou a více agentů do jejich cílů takovým způsobem, aby nedocházelo k jejich kolizím a zablokování. Problém navigace jednoho agenta je řešitelný efektivně a optimálně v polynomiálním čase[2]. Pokud se v prostředí ale vyskytuje více než jeden agent, jedná se o obtížný úkol. Pro takové zadání můžeme požadovat optimální řešení, v takovém případě se ale jedná o NP-úplný problém[14] [28] [20]. Druhým přístupem je hledat nějakou aproximaci optimálního řešení. Takovéto algoritmy často nebývají úplné pro obecné zadání a pro svou efektivitu předpokládají nějaká omezení na vstupní prostředí, které je reprezentováno grafem.

V této práci se zaměříme výhradně na hledání optimálního řešení multi-agentního plánování. Řešení může být optimální v různých vlastnostech v závislosti na míře abstrakce zadání. Můžeme požadovat minimalizaci celkového času řešení, ušlé vzdálenosti všech agentů, počtu kroků agentů, či spotřebované energie při přepravě. V této práci budeme požadovat optimální řešení z pohledu celkového času řešení, tedy doby, která uplynula od začátku do příchodu posledního agenta do jeho cíle. Toto řešení budeme hledat pomocí dobře známého prohledávacího algoritmu A*[7] s využitím konzistentních heuristik. Dále budeme zkoumat řešení multi-agentního plánování pomocí převodu na multi-komoditní toky nad časově expandovaným grafem. Z tohoto převodu odvodíme novou konzistentní heuristiku využitelnou v algoritmu A*. Dále odvodíme a experimentálně ukážeme, v jakých případech tato heuristika dosahuje lepších výsledků než jiné heuristiky.

Jak bylo řečeno, optimální multi-agentní plánování je NP-úplné, to obecně znamená, že jej nedokážeme řešit rychle. Z takového důvodu se často v praxi, když je kladen důraz na rychlost, nevyužívá optimální plánování, ale nějaké suboptimální řešení (například v počítačových hrách). Na druhou stranu motivací pro hledání optimálních cest pro všechny agenty mohou být situace, kdy se prostředí nemění a agenti často vykonávají stejné práce a tedy se často přesouvají po stejných cestách. V takovém případě se vyplatí jednou spočítat optimální trasy a využívat je v budoucnu vícekrát. Příkladem může být pohybování robotů po skladišti. Další motivací je obecně umět řešit NP-úplné problémy rychleji než triviálním přístupem. To může zahrnovat změnu prohledávaného stavového prostoru, lepší prořezávání stavů nebo právě lepší heuristiky, které výpočet navádějí správným směrem.

V rámci celé práce se budeme bavit o agentech, aniž bychom tento pojem formálně definovali. V praxi může být agentem mnoho věcí - robot, vlak, nákladní auto, jednotka v počítačové hře, atd. V naší reprezentaci o agentech nic nemusíme vědět, pouze jejich pozici v každém časovém kroku a jejich kýžený cíl. Neřešíme tedy možnost, že agenti mají své individuální cíle, plány nebo užitkové funkce, jako je tomu v jiných oblastech multi-agentních systémů[25] [24]. Naši agenti budou řízeni centralizovaným způsobem a nemají vlastní "vůli". Také prostředí, které je reprezentováno spojitým grafem, bude značně jednoduché. Agenti

se v něm pohybují sekvenče a všechny hrany mají jednotkovou délku. Přesun libovolného agenta po libovolné hraně tak trvá konstantně stejně a agenti se tedy mohou vyskytovat pouze ve vrcholech v každém časovém okamžiku.

Samotný text je strukturovaný následujícím způsobem. Ve druhé kapitole si formálně definujeme úlohu multiagentního plánování a typy grafů, které k řešení budeme využívat. Ve třetí kapitole se blíže podíváme na související práci. Probereme jiné možné přístupy k řešení problému a také další možná omezení na zadání. Čtvrtá kapitola podrobně popisuje algoritmus A*, jeho úpravu pro naše účely, počítání celkového času řešení, které je potřeba, aby bylo možné ho minimalizovat, a heuristiky, které budeme používat pro porovnání. V páté kapitole se podíváme na převod multi-agentního plánování na multi-komoditní tok. Z tohoto převodu odvodíme naši novou heuristiku a dokážeme některé její vlastnosti. V následující kapitole poté tuto heuristiku experimentálně otestujeme na náhodně generovaných příkladech.

2. Definice

V práci budeme předpokládat základní znalosti z teorie grafů. V této kapitole si přesto definujeme, jak budeme říkat některým konkrétním typům grafů. Dále si také definujeme, co je úlohou multi-agentního plánování.

Definice 1. Orientovaným grafem G *nazveme dvojici* (V,E)*, kde* V *je neprázdná množina vrcholů a množina hran* $E \subseteq V \times V$ *je množina uspořádaných dvojic vrcholů. O hraně* $\langle u, v\rangle$ *potom řekneme, že vede z* u *do* v*. O orientovaném grafu* G *řekneme, že je* silně souvislý*, pokud mezi každými dvěma vrcholy* u *a* v *existuje cesta respektující orientaci hran.*

Definice 2. Čistě orientovaným grafem *budeme nazývat orientovaný graf* $G = (V,E)$*, kde pro každou hranu* $\langle u, v\rangle \in E$ *platí*

$$\langle u, v\rangle \in E \implies \langle v, u\rangle \notin E.$$

Definice 3. *Pro čistě orientovaný graf* $G = (V,E)$ *si zavedeme* vstupní a výstupní stupeň vrcholu *následujícím způsobem:*

$$\text{vstupní stupeň } deg^+(u) = |\{e \in E | \exists v \in V : e = \langle v, u\rangle\}|$$

$$\text{výstupní stupeň } deg^-(u) = |\{e \in E | \exists v \in V : e = \langle u, v\rangle\}|$$

Jak jsme zmínili v úvodu, agenty bereme jako teoretické entity, o nichž nemusíme nic vědět, pouze jejich pozice a kýžené cíle.

Definice 4. *Nechť máme graf* $G = (V,E)$ *a množinu agentů* A*. Pak* stavem grafu *nazveme rozmístění agentů do vrcholů* V *tak, že v každém vrcholu je nejvýše jeden agent. Vrcholy, ve kterých není žádný agent, nazveme* neobsazené vrcholy*. Formálně můžeme stav grafu reprezentovat prostou funkcí*

$$\alpha : A \to V.$$

Pokud naopak potřebujeme vědět, který agent je umístěný v požadovaném vrcholu, můžeme použít inverzní funkci

$$\alpha^{-1} : V \to A \cup \epsilon.$$

Funkce α^-1 je dobře definovaná. To vyplývá z prostoty α a přidání ϵ, do kterého se zobrazí neobsazené vrcholy. Stav grafu v čase i nám dává funkce α_i. Pozici jediného agenta a v čase i budeme značit jako $\alpha_i(a)$.

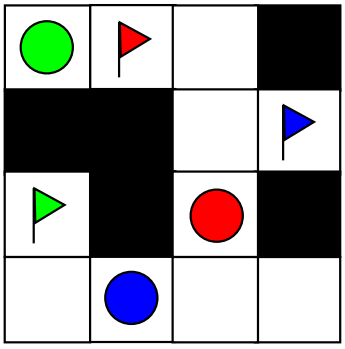
Stav grafu se může změnit na jiný stav krokem agenta. Krok agenta je jeho přesunutí ze stávajícího vrcholu do sousedního neobsazeného vrcholu po orientované hraně. Z časového hlediska můžeme na přesouvání agentů pohlížet dvěma způsoby.

- V každém časovém okamžiku se přesune právě jeden agent ze svého vrcholu do sousedícího neobsazeného vrcholu. V případě těchto *jedno-agentních kroků* se ptáme na součet počtu kroků všech agentů, vedoucích ke splnění úlohy.

- V každém časovém okamžiku provedou právě jeden krok všichni agenti. V tomto případě povolíme jako validní krok i stání na místě. Pro každého agenta musí navíc platit, že se přesouvá do vrcholu, kam se nepřesouvá žádný jiný agent. Jinak by došlo ke kolizi těchto dvou agentů. Pokud se nějaký agent přesune, pak se jiný agent může ve stejném časovém kroku přemístit do tohoto uvolněného vrcholu. Celkovému počtu těchto *multi-agentních kroků* vedoucích ke splnění úlohy budeme říkat *makespan*.

V následující kapitole si popíšeme i jiné omezení na pohyby agentů v rámci makespanu. V rámci této práce budeme ovšem uvažovat tyto podmínky makespanu, pokud nebude výslovně řečeno jinak.

Definice 5. Instance problému multi-agentního plánování v orientovaném grafu *je čtveřice* (G,A,α_0,α_+)*, kde* G *je orientovaný graf,* A *je množina agentů,* α_0 *je počáteční stav a* α_+ *je cílový stav. Úkolem je najít posloupnost kroků agentů, která povede z počátečního do cílového stavu.*



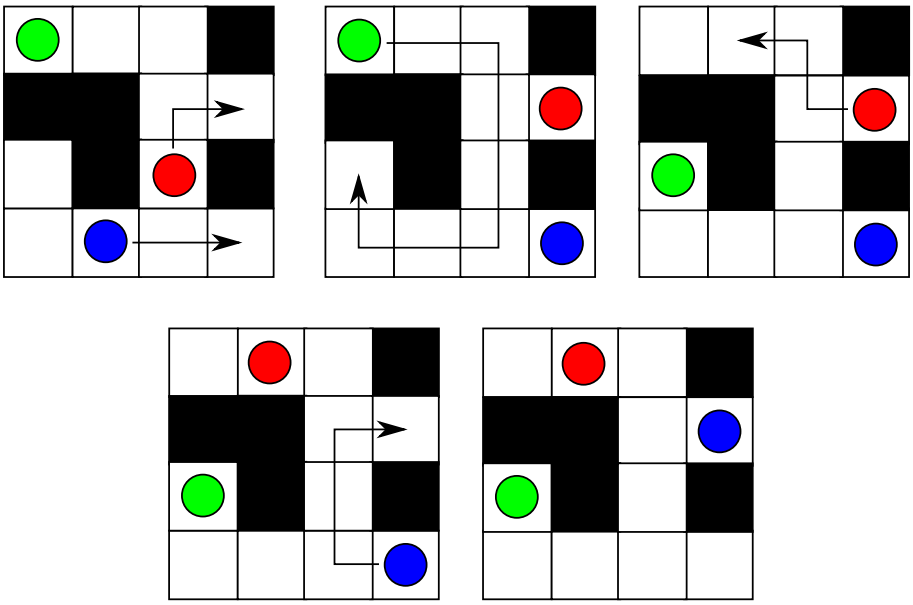
Obrázek 2.1: **Příklad zadání multi-agentního plánování.** Barevné kruhy značí agenty a vlajky značí jejich příslušné cíle. Černé čtverce jsou překážky, tj. vrcholy, do kterých nevedou hrany.

Každý agent má tedy svou výchozí pozici a svůj cíl, kam se snaží dostat (viz Obrázek 2.1). V multi-agentním plánování musíme organizovat všechny agenty tak, aby si navzájem na své cestě nepřekáželi a neblokovali cestu ostatním agentům. Kdyby se v předchozím příkladu každý agent snažil dostat do svého cíle sobecky, nevedlo by to k řešení problému.

V předchozím příkladu si můžeme všimnout, že v navrhovaném řešení (viz Obrázek 2.2) jdou části cest vykonávat zároveň v rámci makespanu. Tedy jako další podmínku úlohy můžeme navíc přidat požadavek na minimalitu počtu kroků vedoucích k řešení. Tato optimální verze je NP-těžký problém[20].

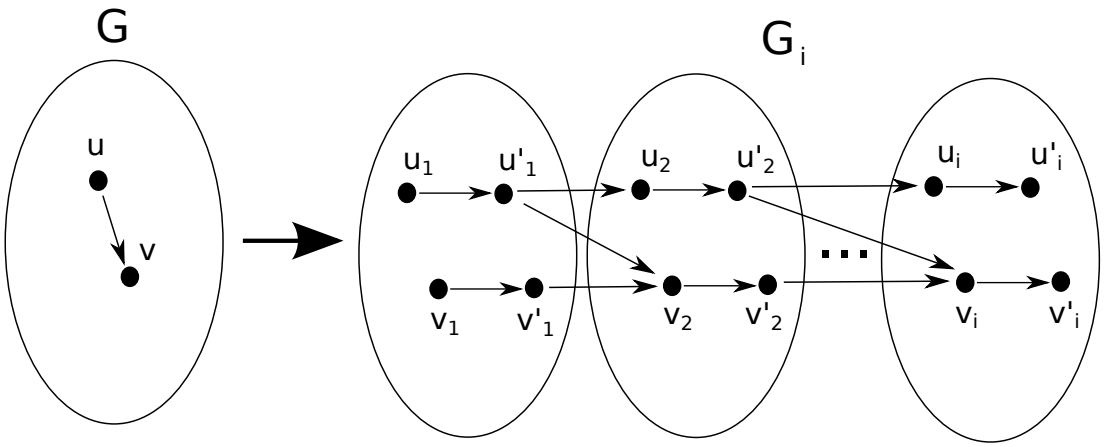
V pozdějších kapitolách budeme využívat časově expandovaných grafů. Tento pojem si zavedeme již nyní a ukážeme si navíc jeho vlastnost, které využijeme.

Definice 6. *Pro graf* $G = (V,E)$ *nazveme* časově expandovaým grafem s i vrstvami *graf* $G_i = (V_i,E_i)$*, kde* V_i *obsahuje vrcholy* $\{v_0, v'_0, \ldots, v_i, v'_i\}$ *pro každý* v *z* V *a* E_i *obsahuje hrany* $\langle u'_j, v_{j+1}\rangle$*, pokud* E *obsahuje hranu* $\langle u, v\rangle$*, pro všechny* $j = 0,\ldots,i-1$*. Navíc přidáme hrany* $\langle u'_j, u_{j+1}\rangle$ *a* $\langle u_j, u'_j\rangle$ *pro všechna* u *a* j*. Všechny vrcholy se stejným indexem budeme nazývat* j*-tou* vrstvou grafu G_i.



Obrázek 2.2: **Možné řešení problému multi-agentního plánování.**

Z definice je vidět, že každá vrstva časově expandovaného grafu obsahuje pouze hrany mezi vrcholem v_j a jeho kopií v'_j. Dále časově expandovaný graf obsahuje pouze orientované hrany z jedné vrstvy do následující vrstvy. Tyto hrany buďto vedou mezi kopiemi vrcholů v'_j a v_{j+1}, nebo odpovídají hranám z původního grafu (viz Obrázek 2.3).



Obrázek 2.3: **Ukázka časově expandovaného grafu.**

Pozorování 1. *Mějme graf G a jeho časově expandovaný graf G_i. Navíc mějme agenta a a jeho stav reprezentovaný funkcí α. Nechť $\alpha(a) = v'_j$ (tj. agent a je ve vrcholu v'_j). Pokud měl tento agent počáteční stav v nějakém vrcholu u_0, pak jeho stav odpovídá stavu v grafu G po nějakých j krocích (včetně stání na místě).*

Důkaz. Posunutí po hraně $\langle u_j, u'_j \rangle$ nepočítáme jako krok agenta. Jinými slovy, když se agent ocitne ve vrcholu u_j, automaticky ho přemístíme do vrcholu u'_j a nezapočítáme mu žádný krok. V každém kroku se agent v grafu G_i musí tedy posunout o jednu vrstvu. Pokud se nachází v j-té vrstvě, musel tedy ujít j kroků.

Mohl se pohybovat po hraně $\langle u'_k, u_{k+1} \rangle$, to odpovídá v G stání na místě. Nebo se mohl pohybovat po hraně $\langle u'_k, v_{k+1} \rangle$, to odpovídá v G přesunutí z vrcholu u do v. Že takováto odpovídající hrana musí být i v původním grafu G, plyne z definice časově expandovaného grafu.

□

3. Související práce

Multi-agentní plánování je rozsáhlé téma, je tedy zřejmé, že existují různé úpravy zadání, omezení a také způsoby řešení. V této kapitole se podíváme na některé dodatečné podmínky, které se vyskytují v různých článcích. Také si stručně popíšeme některá řešení těchto problémů, dříve než se zaměříme na způsob, jakým budeme multi-agentní plánování řešit my.

3.1 Pohyby agentů

Naše dřívější omezení na pohyby agentů není jediné, které se může použít. Další taková omezení si nyní popíšeme. Podle předchozí definice multi-agentního plánování a makespanu mohou agenti vkročit pouze do volného vrcholu. To zahrnuje i vrcholy, které v tomto časovém kroku opustil jiný agent. Kromě obecných kroků, kdy si agenti nijak nepřekážejí, tedy dovolujeme i pohyb skupiny agentů naráz po nějaké cestě v grafu. V Obrázku 3.1 toto odpovídá možnosti b).

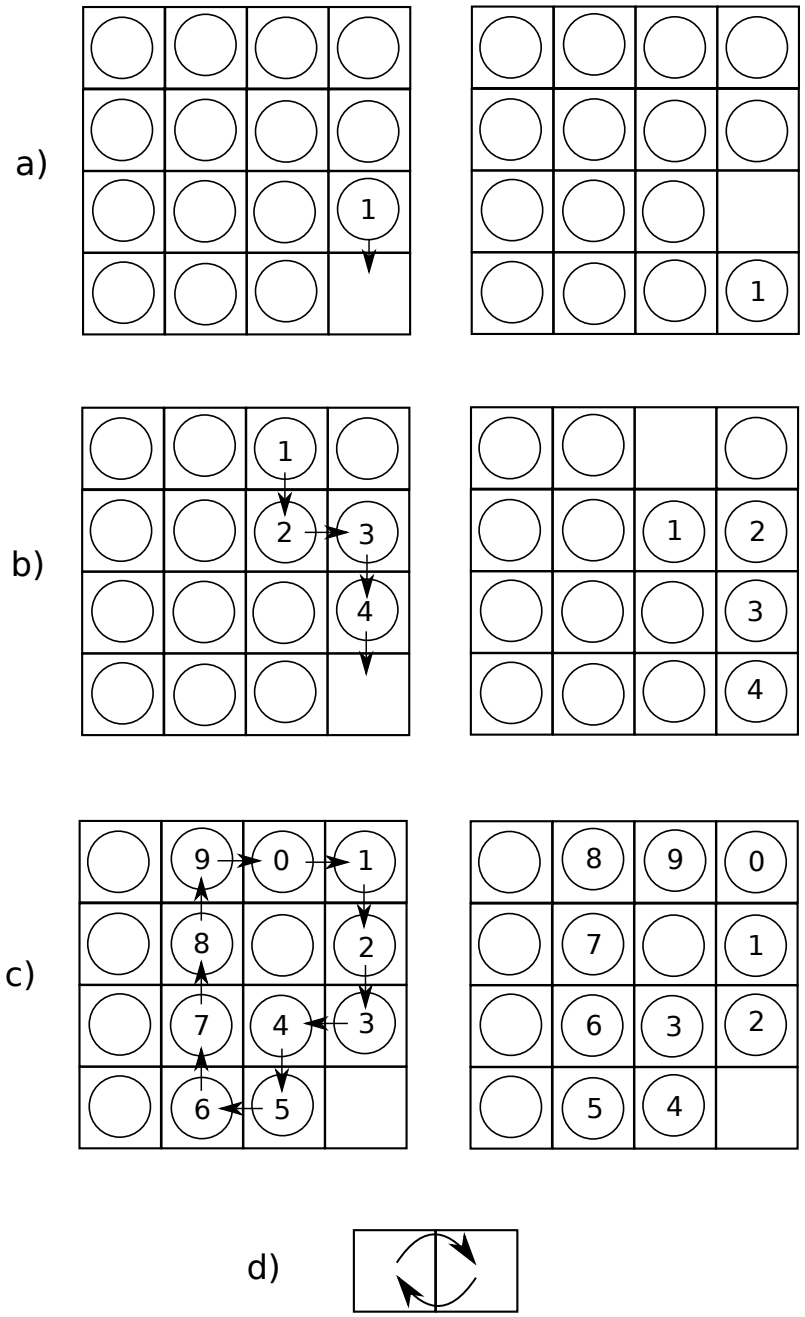
Další možností je, že agenti se mohou pohybovat pouze do volných vrcholů a nemohou využívat vrcholy, které jiný agent právě opustil. Tímto omezením zabráníme pohybu více agentů v řadě, jako tomu bylo v předchozím případě. Takováto definice odpovídá pohybům v problému Loydovy patnáctky. Na Obrázku 3.1 můžeme vidět příklad takového pohybu v možnosti a). Pokud hledáme optimální řešení multi-agentního plánování za použití takovýchto kroků, je problém také NP-úplný[14].

Poslední možnost, kterou si představíme, je, že agenti se mohou pohybovat nejen do volných vrcholů, které právě opouští jiný agent, jako tomu je v naší původní definici, ale navíc se mohou pohybovat i po plně obsazených kružnicích. Pokud v grafu existuje nějaká orientovaná kružnice délky alespoň tři, která je plně obsazená agenty, pak se po této kružnici mohou agenti pohnout v jednom směru. V Obrázku 3.1 toto odpovídá možnosti c). Pokud navíc hledáme optimální řešení, pak i tento problém zůstává NP-úplný [28]. Této možnosti budeme říkat *multi-agentní plánování s rotacemi*. V pozdějších kapitolách jej využijeme pro převod na jiný NP-úplný prolém a konstrukci heuristiky. Můžeme si všimnout, že v žádném z uvedených omezení nedovolujeme dvěma sousedícím agentům prohození, které v Obrázku 3.1 představuje možnost d).

Použití různého omezení na pohyb agentů může mít vliv nejen na délku makespanu, ale i na řešitelnost celé úlohy. První dvě možnosti jsou zajisté ekvivalentní, co se řešitelnosti týče. Agenti se mohou pohybovat po cestě postupně, a tak docílí stejného stavu pouze za více kroků. Multi-agentní plánování s rotacemi ale může řešit i jinak nesplnitelné problémy. Triviálním příkladem je mřížka bez volného vrcholu.

3.2 Optimalita řešení

Jak bylo již zmíněno, optimální řešení je NP-úplné. To má za důsledek velké časové nároky pro výpočet. Často je ovšem zapotřebí najít nějaké řešení rychle i za cenu, že se nejedná o nejlepší řešení. Například v real time strategických hrách



Obrázek 3.1: **Ukázka různých druhů pohybů agentů.** a) V každém časovém okamžiku se mohou agenti pohnout pouze do volného vrcholu. b) Agenti se mohou v jednom kroku pohnout po cestě, pokud je na konci cesty volný vrchol. c) Agenti se mohou pohybovat v jednom směru po libovolné plně obsazené kružnici. d) Prohození sousedních agentů není povolené.

je potřeba naplánovat cestu pro velké množství jednotek pohybujících se naráz. Plánování pomocí prohledávání stavového prostoru (například pomocí algoritmu A*, který si představíme v následující kapitole) je sice úplné a optimální, je ovšem pomalejší než nalezení suboptimálního řešení.

Dalším optimálním řešením je algoritmus Conflict Based Search (CBS)[16]. Tento algoritmus je rozdělený na vyšší úroveň, kde se staví a prohledává binární strom omezení, a na nižší úroveň, kde se hledá cesta pro jednotlivé agenty zvlášť

na základě daných omezení. V binárním stromu každý vrchol obsahuje seznam omezení, seznam cest pro všechny agenty a celkovou cenu řešení. Vrchol se označí jako cílový, pokud žádná z cest v něm neobsahuje konflikt (tj. žádná dvojice agentů se nevyskytuje ve stejný čas ve stejném vrcholu). Prohledávání probíhá podle nejlepší ceny řešení. Při prozkoumávání vrcholu proběhne nalezení cest pro každého agenta jednotlivě s omezeními daného vrcholu. Tyto cesty jsou poté validovány, a pokud se nalezne nějaký konflikt, je jasné, že v nějakém čase i se ve vrcholu v vyskytují naráz agenti a_l a a_k. Vytvoří se tak dva nové vrcholy, do kterých se přidají všechny podmínky z rodiče, a navíc do jednoho podmínka, která zakazuje agentu a_k být ve vrcholu v v čase i, a do druhého obdobná podmínka pro agenta a_l.

Obecně se dá plánování řešit dvěma možnými způsoby [8] [6]. Prvním je centralizované řešení, kdy existuje jediný rozhodovací prvek, který plánuje naráz cestu všem agentům. Tento přístup může být jak optimální, tak neoptimální. Druhým řešením je decentralizovaný přístup, kdy každý agent naplánuje svou cestu bez ohledu na pozice a plány ostatních agentů. V případě, že by mělo dojít ke kolizi, jsou plány agentů nějakým způsobem upraveny. Tento druhý přístup obecně nemůže nalézt optimální řešení. Příkladem takového algoritmu může být Local Repair A*[18], kdy každý agent plní svůj plán podle předem nalezené nejkratší cesty. V případě, že se chce přesunout na pozici, která je blokována jiným agentem, přeplánuje svou trasu s touto nově známou překážkou. Tyto algoritmy často bývají neúplné a v obecném případě tak nezaručují řešení. Konkrétně Local Repair A* má množství nedostatků v obtížných mapách s velkým počtem agentů a zúžení [13]. Existují ale vylepšení těchto algoritmů, které nacházejí úspěšnější cesty[17] pomocí rezervací plánů v časovém rozvrhu.

Další možností je zaměřit se na konkrétní typy problémů, kdy klademe nějaké požadavky na vstupní graf, případně i počet agentů. Například pro mřížkové grafy, které jsou často využívány ve hrách, logistice, organizaci skaldiště a další, existují specializované algoritmy. Jedním takovým příkladem je Multi-Agent Path Planing (MAPP) [23], který opět plánuje cesty všem agentům jednotlivě a případné kolize řeší obcházením agentů nebo uhýbáním stojících agentů. Posledním typem grafů, které zmíníme, jsou orientované 2-souvislé grafy. Tyto grafy mají vlastnost, že jdou zkonstruovat z kružnice, ke které se přidávají uši. Algoritmus BIBOX [19] využívá této vlastnosti dekompozicí grafu na kružnici a přidané uši a následným seřazením agentů v kružnici a dále iterativním umístěním agentů v jednotlivých uších. Nejprve se vyřeší poslední přidané ucho, tím vznikne stejný problém, ovšem menších rozměrů. Nakonec nám zůstane pouze původní kružnice, která již ovšem obsahuje agenty ve správné permutaci, stačí je tedy pootočit podél této kružnice. Navíc pokud takovýto graf má alespoň dva neobsazené vrcholy, je vždy problém splnitelný.

3.3 Týmy agentů

Zajímavým zobecněním multi-agentního plánování je rozdělení agentů do skupin. Mějme množinu agentů A a množinu cílových vrcholů C. Pak tyto množiny rozdělíme na k podmnožin A_i a C_i tak, že $|A_i| = |C_i|$ pro $i = 1, \ldots, k$. Úkolem je navigovat všechny agenty do příslušných cílů tak, že libovolný agent z A_i skončí v libovolném vrcholu z C_i pro všechna i. Toto zobecnění má dva extrémy. Prv-

ním je, že každá z množin je velikosti právě jedna. Pak se jedná o multi-agentní plánování tak, jak jsme si je definovali. Druhým extrémem je, když máme pouze jednu podmnožinu velikosti původní A. Pak se libovolný agent musí dostat do libovolného cílového vrcholu. Pokud v tomto případě uvažujeme multi-agentní plánování s rotacemi, lze optimální řešení najít v polynomiálním čase [26]. To hledáme na časově expandovaném grafu pomocí toku. Obdobně obecnou variantu s více skupinami, která je v optimální verzi NP-úplná, můžeme řešit pomocí multi-komoditního toku na expandovaném grafu. Toto téma blíže prozkoumáme v jedné z následujících kapitol. Jedná se totiž o základní myšlenku při konstrukci naší nové heuristiky.

4. Řešení pomocí A*

4.1 Algoritmus A*

V této práci se budeme výhradně zabývat nalezením optimálního řešení úlohy multiagentního plánování. K tomuto účelu budeme využívat algoritmus A* (A star), který byl poprvé publikován v roce 1968[7]. Nyní si představíme jeho základní verzi pro hledání cesty v grafu a dokážeme některé jeho vlastnosti. A* používá hladový přístup pro hledání optimální cesty z počátečního do koncového vrcholu. V průběhu výpočtu navštívené vrcholy označujeme jako otevřené nebo uzavřené. Na začátku je počáteční vrchol označený jako otevřený a všechny ostatní nejsou označené. Otevřené vrcholy jsou udržovány v prioritní frontě, která je setříděna na základě funkce $f(v) = g(v)+h(v)$. Kde $g(v)$ je již známá cena cesty z počátečního vrcholu do vrcholu v a $h(v)$ je heuristický odhad ceny z v do koncového vrcholu. Algoritmus vždy vybere z prioritní fronty vrchol v s nejnižší $f(v)$, zkontroluje, zda se jedná o cílový vrchol, a pokud ne, najde jeho sousedy, které přidá k otevřeným vrcholům. Vybraný vrchol je poté označený jako uzavřený. Dobré vlastnosti tohoto algoritmu jsou závislé na použité heuristice h. Můžeme si všimnout, že pokud použijeme nulovou heuristiku ($h(v) = 0$ pro každý vrchol), pak je algoritmus ekvivalentní Dijkstrovu algoritmu.

Definice 7. *O heuristice $h(v)$ řekneme, že je* přípustná*, pokud se pro všechna v jedná o spodní odhad nejkratší cesty do koncového vrcholu.*

Definice 8. *O heuristice $h(v)$ řekneme, že je* konzistentní*, pokud splňuje následující nerovnici*

$$h(v) \leq c(v,u) + h(u),$$

kde $c(v,u)$ je cena cesty z v do u.

Konzistentní heuristika je vždy přípustná, opačný vztah však obecně nemusí platit[15].

Věta 2. *Pokud použijeme konzistentní heuristiku, pak nalezená cesta pomocí A* je optimální a navíc do uzavřených vrcholů jsme již našli optimální cestu.*

Důkaz. Konzistentní heuristika nám zaručuje, že hodnota $f(n)$ se během žádné cesty nesníží. Mějme zpracovávaný vrchol n s $f(n)$. To znamená, že všechna ostatní $f(m)$ otevřených vrcholů nejsou menší. Tedy ze všech cest přes otevřené vrcholy neexistuje kratší cesta do n. Pokud je tedy zpracovávané n cílový vrchol, nalezli jsme optimální cestu. Zároveň do libovolného uzavřeného vrcholu neexistuje lepší než již nalezená cesta.

□

Pokud používáme konzistentní heuristiku, pak máme díky předchozí větě zaručeno, že do uzavřených vrcholů již nepovede lepší cesta, tedy se k nim během výpočtu nemusíme vracet. V obecném případě musíme ovšem kontrolovat, zda jsme nenalezli lepší cestu do nějakého z otevřených vrcholů (viz Algoritmus 1).

Algorithm 1 Algoritmus A*

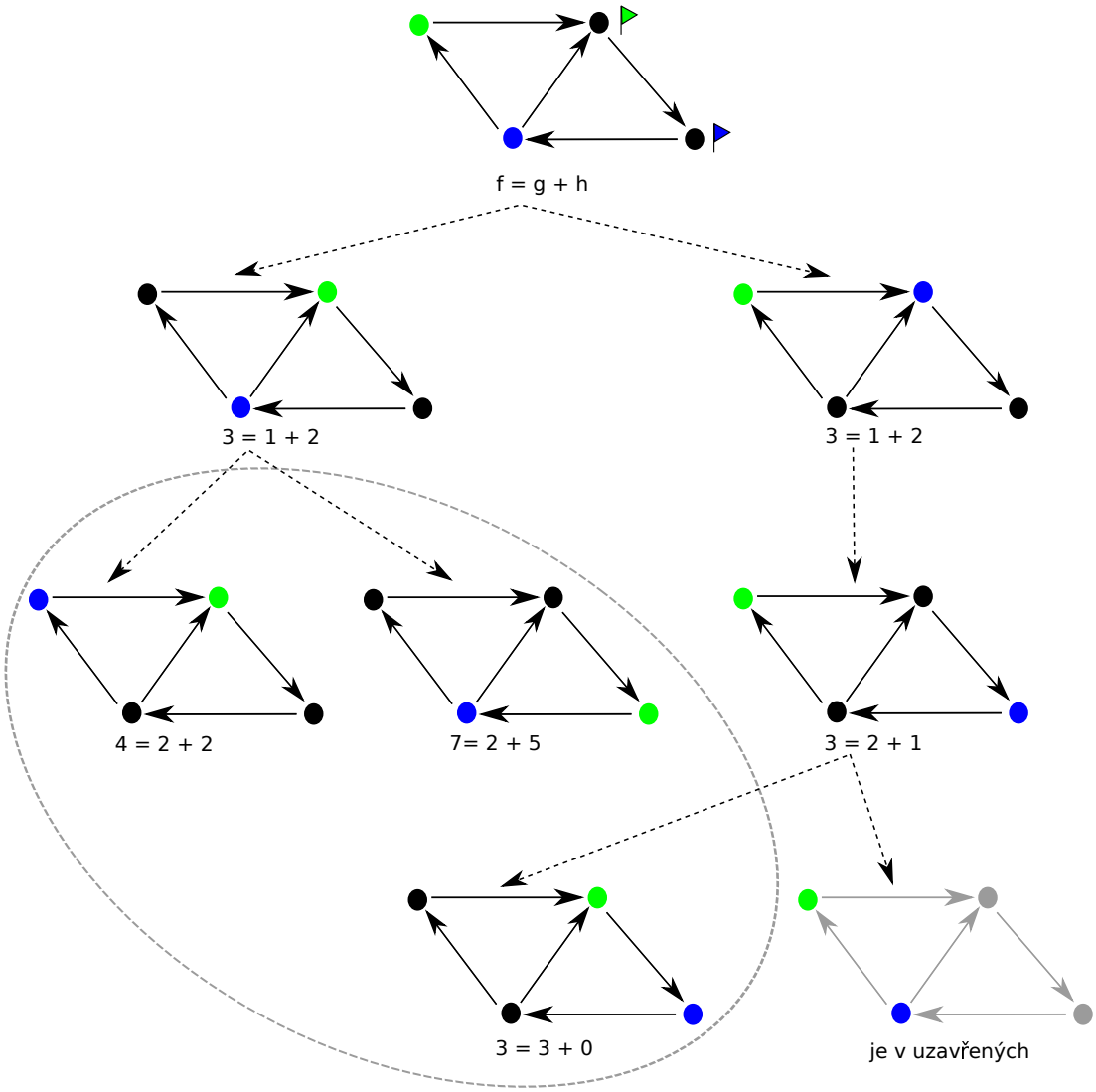
```
1: procedure AStar
2:     closed ← {}
3:     open ← {start}
4:     while open not empty do
5:         current ← extractMinimalF(open)
6:         addToClosed(current)
7:         if current = goal then return success
8:         for each neighbor of current do
9:             if neighbor in closed then continue
10:            if neighbor in open then
11:                keepBetter()
12:                continue
13:            addToOpen(neighbor)
14:    return failure
```

Takto popsaný algoritmus je vhodný pro hledání cest mezi dvěma vrcholy, pokud máme dostupnou nějakou heuristiku. Uvažujme příklad hledání nejkratší cesty mezi dvěma městy - výchozím a cílovým. Známe silniční síť, která spojuje více měst, a také máme k dispozici vzdušnou vzdálenost všech měst od cílového města, ta bude sloužit jako heuristika.

Tento přímočarý postup ovšem nestačí pro náš problém, kdy chceme navigovat více agentů do více cílů. Místo toho budeme prohledávat strom stavů, který zároveň budujeme během jeho prohledávání. Vrcholy tohoto stromu odpovídají stavům grafu tak, jak jsme je definovali v Definici 4. Každý vrchol je navíc označený jako otevřený nebo uzavřený. Na začátku je otevřený pouze kořen, který odpovídá stavu α_0 (řádek 3 v Algoritmu 1). Při uzavírání vrcholu v s nejnižší hodnotou funkce f zkoumáme všechny stavy, do kterých se dá dostat ze stavu odpovídajícímu vrcholu v. Nové stavy tvoříme postupným přesouváním jednotlivých agentů do všech možných sousedních vrcholů (řádek 8). Pokud je tento nový stav n již přítomný ve vybudované části stromu, ale odpovídá uzavřenému vrcholu, zajisté do něj již známe optimání cestu, a tedy tento stav nebudeme přidávat a pokračujeme dále (řádek 9). Jinak vrcholu v přidáme nového syna v_1, který odpovídá stavu n a označíme v_1 za otevřený. Pokud je n shodný se stavem odpovídajícím nějakému otevřenému vrcholu l, pak zkontrolujeme, zda jsme nenalezli lepší cestu do tohoto stavu. Pokud je cesta od kořene stromu do l lepší než cesta od kořene stromu do v_1, tj. $g(l) \leq g(v_1)$, pak smažeme vrchol v_1. Naopak pokud platí $g(l) > g(v_1)$, pak smažeme vrchol l a pokračujeme na další stav (řádek 10-12). Pokud nový stav n nebyl přítomný ve vybudované části stromu, pak pouze označíme nový vrchol v_1 jako otevřený (řádek 13). Výpočet končí, pokud uzavíráme vrchol, který odpovídá koncovému stavu α_+ (řádek 7). Pokud nám již nezbývá žádný otevřený vrchol a stále jsme nenalezli koncový stav, je úloha neřešitelná (řádek 4).

Díky krokům na řádcích 10-12, kdy odstraňujeme horší cesty do jednotlivých stavů, pak si pro každý unikátní navštívený stav pamatujeme jednoznačnou cestu. Stavěný graf je tedy opravdu stromem. Dále si můžeme všimnout, že kořenem stromu je počáteční stav α_0 a otevřené vrcholy mohou být pouze listy stromu,

protože k vrcholu přidáváme potomky, pouze když ho zpracováváme, a po zpracování je vrchol vždy uzavřen. Vrchol může být listem stromu a zároveň uzavřený, pokud jsme k němu již při jeho uzavírání nevytvořili žádné potomky. Tedy z něj nevedly žádné přípustné stavy nebo tyto stavy byly již uzavřené nebo otevřené s lepší cestou. Případně v průběhu výpočtu tento vrchol přišel o všechny své potomky, protože jsme do nich nalezli lepší cestu. Příklad stavěného stromu lze vidět na Obrázku 4.1.



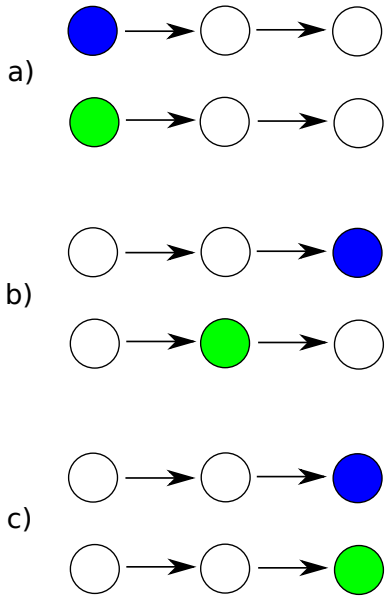
Obrázek 4.1: **Příklad budovaného stavového stromu.** Kořen je počáteční stav s vyznačenými koncovými pozicemi. Zakroužkované stavy jsou právě otevřené stavy. V příštím kroku se vybere otevřený stav s nejmenší hodnotou funkce g, což je koncový stav.

4.2 Hodnota funkce g

Nyní si popíšeme, jak přesně počítáme hodnotu funkce g. Jak bylo zmíněno dříve, můžeme se na přesouvání agentů dívat dvěma možnými způsoby. Prvním, přímočařejším postupem je, že v jednom časovém okamžiku se přesune právě

jeden agent do nějakého volného sousedícího vrcholu. Tuto variantu budeme nyní značit jako g_s. Výpočet hodnoty g_s během řešení úlohy je jednoduchý ze struktury stavěného stromu. Kořen r odpovídající stavu α_0 má $g_s(r) = 0$. Pro libovolný vrchol v_1 s otcem v platí $g_s(v_1) = g_s(v) + 1$. To vyplývá z toho, že nový stav vytvoříme přesunutím právě jednoho agenta do volného sousedního vrcholu, jak bylo ilustrováno na Obrázku 4.1. Výhodou, která souvisí se snadným výpočtem, je fakt, že všechny vrcholy v jedné úrovni stavového stromu mají stejnou hodnotu g_s. Hodnota g_s je tedy zaměnitelná s vrstvou stavového stromu. Pro každý nově vytvořený vrchol ve stavovém stromu zároveň spočítáme i heuristiku.

Druhým přístupem je makespan a tuto variantu budeme značit jako g_m. V tomto případě se mohou pohnout všichni agenti najednou. I s tímto přístupem stavíme strom stejným způsobem, tedy nové stavy získáváme přesunutím jednoho agenta. Takto nezachováme hezkou souvislost hladiny stromu a hodnody g_m, ale vyvarujeme se tak jiným problémům. Kdybychom totiž chtěli tuto vlastnost zachovat, tak pro každý stav musíme jako potomky vytvořit všechny možné podmnožiny agentů, kteří se přesunuli všemi možnými směry. To by vedlo k obrovskému větvení stromu. Je tedy lehčí nové stavy tvořit přesunutím jednoho agenta a zpětně dopočítat, zda jsme se tímto krokem dostali do nové hodnoty makespanu, tj., zda pro vrchol v_1 s otcem v platí $g_m(v_1) = g_m(v)+1$. Hodnotu g_m zvýšíme, pokud se posunul agent, který se již v této hodnotě makespanu posouval. Heuristiku v tomto případě počítáme pouze, pokud dojde ke zvětšení hodnoty g_m. Pokud ne, dostane vrchol hodnotu heuristiky stejnou, jako má jeho otec. Toto děláme z toho důvodu, že v jednom časovém okamžiku mají provést krok nebo stát na místě všichni agenti a heuristiku počítáme pouze pro nový časový krok. Naše implementace, kdy vytváříme nový vrchol ve stavovém stromu po přesunutí jednoho agenta, je pouhé usnadnění reprezentace.



Obrázek 4.2: **Pořadí přesunutí agentů v rámci** g_s **a** g_m a) Počáteční stav b) Stav, do kterého se lze dostat dvěma různými způsoby c) Koncový stav

Je vhodné nahlédnout, že v případě makespanu záleží na pořadí, v jakém se agenti přesouvají. Uvažujme graf a rozestavení agentů z Obrázku 4.2. Vždy

pohneme jedním agentem a zpětně dopočítáme, zda se nezvětšila hodnota g_m. Do stavu b) se lze dostat dvěma různými způsoby:

1. Nejprve dvakrát pohneme modrým agentem a poté jednou zeleným. V takovém případě je $g_m = 2$.

2. Nejprve pohneme modrým agentem, poté jednou zeleným a nakonec jednou modrým. V takovém případě je $g_m = 2$.

Oba způsoby nám dají stejnou hodnotu g_m, zapamatujeme si tedy pouze jeden z nich. První z nich nám ale dá celkové řešení v $g_m = 3$, zatímco druhý dá optimální řešení $g_m = 2$. Z tohoto důvodu navíc ke stavům přidáme indikátory pro všechny agenty. Ty udávají, zda se daný agent v daném čase již přesouval či nikoliv. To nám sice do stavěného stromu přidá duplikátní vrcholy, ty ale od sebe dokážeme rozlišit pomocí zmíněných indikátorů a vyhneme se tak problémům, kdy se do jednoho stavu můžeme dostat dvěma různými zůsoby, ale zapamatujeme si jen jeden, ten neoptimální. V případě g_s tyto indikátory nepotřebujeme, protože počet kroků nezáleží na pořadí, v jakém se provedly.

Rozdílné funkce g uvádíme, protože různé heuristiky dávají odhady buď pro g_s, nebo pro g_m. Dvě jednoduché heuristiky si rozebereme v následující podkapitole a v následující kapitole se blíže podíváme na námi navrženou novou heuristiku pro g_m.

4.3 Heuristiky

První jednoduchou heuristikou je suma nejkratších cest. Pro stav α_i máme heuristiku

$$\sum_{a \in A} d(\alpha_i(a), \alpha_+(a)),$$

kde $d(\alpha_i(a), \alpha_+(a))$ je vzdálenost mezi vrcholem, ve kterém je ve stavu i agent a, a cílem agenta a. Tato heuristika odhaduje počet kroků. Jistě totiž každý agent musí ujít alespoň tolik, aby se dostal do svého cíle. Tyto cesty se mohou pouze prodloužit, pokud dojde ke kolizi dvou nebo více agentů, kteří se musí navzájem obejít. Z toho plyne i následující pozorování.

Pozorování 3. *Heuristika sumy nejkratších cest je spodním odhadem celkového počtu kroků, a tedy přípustná pro g_s.*

Pro použití heuristiky v algoritmu A*, tak jak jsme jej definovali, musíme navíc ukázat i její konzistenci.

Pozorování 4. *Heuristika sumy nejkratších cest je konzistentní pro g_s.*

Důkaz. Konzistentní heuristika musí splňovat vztah

$$h(n) \leq c(n,k) + h(k), \qquad (*)$$

pro všechny stavy n a jeho potomky k. $c(n,k)$ značí skutečnou cenu přechodu ze stavu n do k. V našem použitém značení platí vztah $c(n,k) = g_s(k) - g_s(n)$.

Vztah (*) stačí dokázat pouze pro n a k, kde k je potomek stavu n [12]. V takovém případě vždy platí $c(n,k) = 1$, protože se posune právě jeden agent a tím zvýší g_s o jedna. $h(k)$ může vzrůst nebo zůstat stejné, pokud se agent posune směrem od svého cíle. Na druhou stranu, pokud se posune směrem ke svému cíli po nejkratší cestě, může klesnout maximálně o jedna.

□

Druhou heuristikou je nejdelší z nejkratších cest všech agentů do cíle. Pro stav α_i máme heuristiku

$$\max_{a \in A} d(\alpha_i(a), \alpha_+(a)),$$

kde $d(\alpha_i(a), \alpha_+(a))$ je opět vzdálenost mezi vrcholem, ve kterém je ve stavu i agent a, a cílem agenta a. Tato heuristika odhaduje makespan potřebný k dosažení cíle z daného stavu. Jistě totiž agent, který má před sebou nejdelší cestu, ji musí ujít celou, ale může se přesunout pouze jednou v každém časovém okamžiku. Cesty ostatních agentů neuvažujeme, protože se agenti přesouvají naráz. Tento čas se opět může pouze prodloužit, pokud bude nutné vyřešit kolize agentů, a tím dostáváme následující tvrzení.

Pozorování 5. *Heuristika nejdelší z nejkratších cest je spodním odhadem makespanu, a tedy přípustná pro g_m.*

Pozorování 6. *Heuristika nejdelší z nejkratších cest je konzistentní pro g_m.*

Důkaz. Podobně jako v předchozím případě chceme ukázat, že platí vztah

$$h(n) \leq c(n,k) + h(k),$$

pro všechny stavy n a jeho potomky k. V jednom časovém okamžiku provedou krok všichni agenti (platným krokem je i stání na místě) a tím se dostaneme do nového stavu. Opět tedy vždy platí $c(n,k) = 1$. $h(k)$ může vzrůst nebo zůstat stejné, pokud se agent neposune nebo posune směrem od svého cíle. Na druhou stranu, pokud se posune směrem ke svému cíli po nejkratší cestě, může klesnout maximálně o jedna.

□

Obě tyto zmíněné heuristiky mají výhodu ve svém rychlém výpočtu. Pokud totiž máme k dispozici tabulku vzdáleností mezi všemi vrcholy v grafu, pak nám pro výpočet stačí pouze vyčíst požadovanou hodnotu pro každého agenta.

5. Heuristika pomocí toku

V této kapitole se podíváme na převod multi-agentního plánování na hledání nejlevnějšího multi-komoditního toku v síti. Z tohoto převodu relaxací multi-komoditního toku na jedno-komoditní tok odvodíme námi navrhovanou heuristiku a dokážeme její vlastnosti. Heuristiku budeme nejprve konstruovat pro čistě orientované grafy, později ale ukážeme, že v nezměněné podobě bude dávat stejné hodnoty i pro neorientované grafy.

5.1 Souvislost s multikomoditním tokem

Nejprve definujeme co je multikomoditní tok, se kterým budeme dále pracovat.

Definice 9. *Mějme tokovou síť $G = (V,E)$, kde každá hrana $(u,v) \in E$ má přiřazenu celočíselnou kapacitu $cap(u,v)$ a celočíselnou cenu $cost(u,v)$. Navíc máme k komodit $K_1, K_2, \dots K_k$ definovaných jako $K_i = (s_i, t_i, d_i)$, kde s_i je zdroj, t_i je stok a d_i je poptávka dané komodity. Tok komodity i po hraně (u,v) je $f_i(u,v)$. Pak* multi-komoditní tok *musí splňovat následující podmínky:*

1. *Podmínky kapacity*
$$\sum_{i=1}^{k} f_i(u,v) \leq cap(u,v)$$
2. *Kirchhoffův zákon*
$$\sum_{w \in V} f_i(u,w) = 0$$
pro $u \neq s_i, t_i$
3. *Splnění poptávek*
$$\sum_{w \in V} f_i(s_i,w) = \sum_{w \in V} f_i(w,t_i) = d_i$$

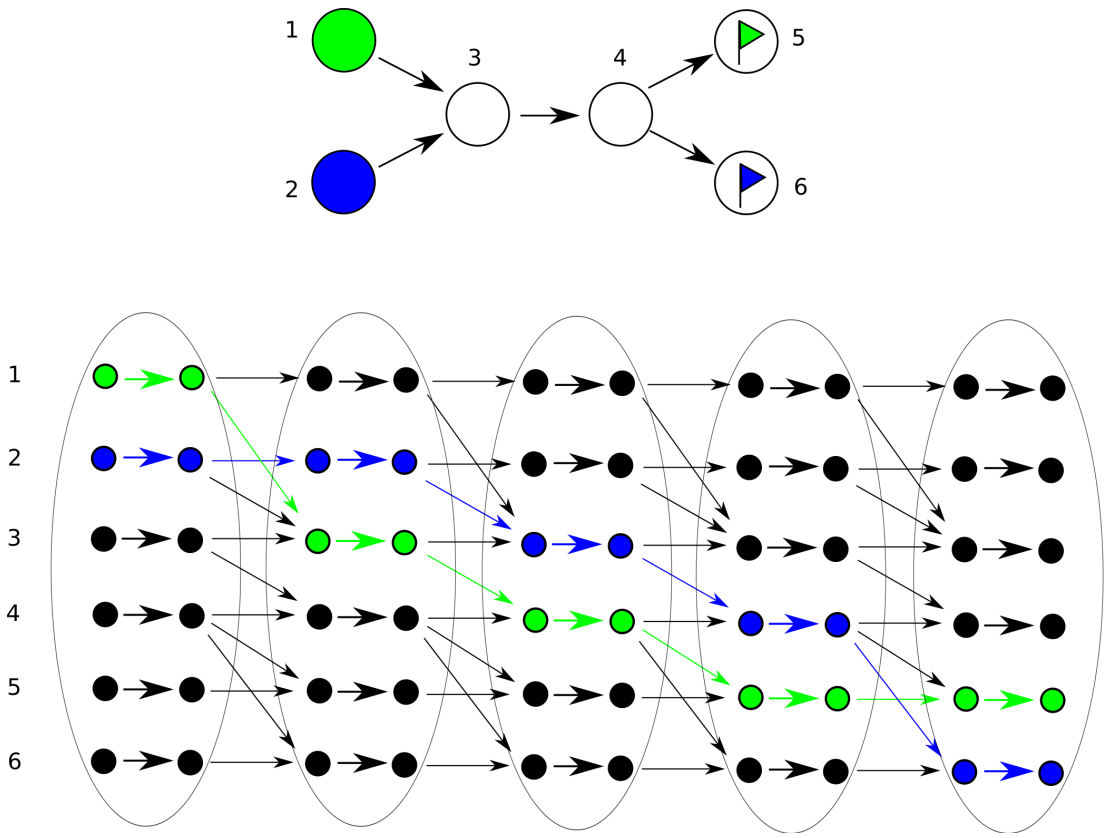
Pro takto zadefinovaný tok poté chceme hledat nejlevnější multi-komoditní tok, tedy minimalizujeme sumu

$$\sum_{(u,v) \in E} (cost(u,v) \sum_{i=1}^{k} f_i(u,v)).$$

Tento tok stále musí splňovat výše uvedené podmínky, tedy nelze triviálně zvolit nulový tok, pokud máme pro komodity nenulové poptávky. Tento problém je již pro dvě komodity NP-úplný[4].

Pokud se díváme na agenty jako komodity, můžeme tyto dva problémy spojit využitím časově expandovaného grafu z Definice 6. V následujícím převodu předpokládáme, že se jedná o multi-agentní plánování s rotacemi představené v kapitole 3.

Nechť má problém řešení v makespanu n, pak vytvoříme ke vstupnímu grafu G časově expandovaný graf G_{n+1}. Ke každému agentu a_i přiřadíme jednu komoditu



Obrázek 5.1: **Řešení multi-agentního plánování pomocí multi-komoditního toku.** Nahoře je vidět původní graf s agenty a vlajkami, které reprezentují jejich cíle. Ve spodní části je řešení tohoto zadání pomocí multi-komoditního toku na expadnovaném grafu. Kopie vrcholů jsou označeny čísly, která odpovídají původnímu grafu. Barvy hran odpovídají nasyceným cestám jednotlivých komodit. Zdroj „zelené“ komodity je ve vrcholu 1 v první vrstvě, její stok je ve vrcholu 5 v poslední vrstvě. Zdroj „modré“ komodity je ve vrcholu 2 v první vrstvě, její stok je ve vrcholu 6 v poslední vrstvě.

K_i, která bude mít zdroj v 0-té vrstvě G_{n+1}, ve vrcholu, který odpovídá počátečnímu stavu agenta a_i. Stok se bude nacházet v $n+1$-té vrstvě G_{n+1}, ve vrcholu, který odpovídá koncovému stavu a_i. Všem hranám v grafu přiřadíme jednotkové kapacity. Hranám $\langle u'_j, v_{j+1} \rangle$ přiřadíme cenu 1 a hranám $\langle u_j, u'_j \rangle$ přiřadíme nulovou cenu. Jinými slovy, hrany mezi vrstvami mají jednotkové ceny a hrany v rámci jedné vrstvy mají nulové ceny. Každé komoditě i přiřadíme požadavek $d_i = 1$.

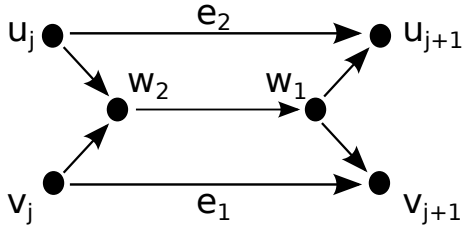
Pokud nyní nalezneme v takovémto grafu nejlevnější multi-komoditní tok, pak tyto toky pro každou komoditu odpovídají cestám agentů v původním grafu. Pokud je nasycená hrana $\langle u_j, u_{j+1} \rangle$, odpovídá to situaci, že agent v časovém kroku j stál na místě. Pokud je naopak nasycena hrana $\langle u_j, v_{j+1} \rangle$, odpovídá to přechodu agenta z vrcholu u do vrcholu v v čase j. Ceny hran se tak platí pouze za hrany, které odpovídají časovému kroku agenta v původním grafu, ať už se agent pohybuje nebo stojí na místě. Můžeme si všimnout, že dva agenti nikdy nemohou být v jednom vrcholu díky rozdělení vnitřních vrstev časově expandovaného grafu. Pokud je navíc makespan n optimální, nalezli jsme i optimální řešení. Toto optimální n můžeme hledat iterativně. To vyplývá z následujícího tvrzení, které

v této práci uvádíme bez důkazu. Důkaz je publikován v článku Planning optimal paths for multiple robots on graphs autorů Jingjin Yu a Steven M. LaValle[27].

Tvrzení 7. *Existuje bijekce mezi řešeními multi-agentního plánování, které je omezené makespanem T, a řešeními multi-komoditního toku nad odpovídajícím časově expandovaným grafem s T+1 vrstvami.*

Příklad řešení takového převodu můžeme vidět na obrázku 5.1. Krokům odpovídají hrany mezi sousedícími vrstvami, proto je pro tento příklad potřeba vytvořit 5 vrstev, i když problém má řešení v makespanu 4. Hrany uvnitř vrstev jsou přítomné z důvodu zabránění nedovoleného pohybu agentů.

Je důležité si povšimnout, že takto navržený časově expandovaný graf dává korektní řešení pouze pro čistě orientované grafy. Pokud by ve vstupním grafu existovala neorientovaná hrana, pak by mohlo dojít k zakázanému prohození agentů. K řešení tohoto problému lze využít složitější grafovou konstrukci, která se vytvoří pro každou hranu $\langle u, v\rangle$ z původního grafu [26] (viz Obrázek 5.2). Hrany e_1 a e_2 opět slouží pro simulování agenta, který stojí na místě. Přidané vrcholy w_1 a w_2 slouží pro zaručení, že nalezený tok bude odpovídat pouze pohybu ve směru $\langle u, v\rangle$ nebo $\langle v, u\rangle$. Všechny přidané hrany mají opět jednotkovou kapacitu. Ceny horizontálních hran ($\langle u_j, u_{j+1}\rangle, \langle v_j, v_{j+1}\rangle, \langle w_2, w_1\rangle$) jsou 1 a zbytek má cenu 0. To opět odpovídá tomu, že cena se platí pouze za hrany, které odpovídají kroku agenta v původním grafu.



Obrázek 5.2: **Konstrukce mezi vrstvami časově expandovaného grafu pro neorientované grafy.** Pro každou hranu $\langle u, v\rangle$ vytvoříme takovouto konstrukci, která zamezí prohození agentů.

Oba druhy časově expandovaných grafů jsou značně větší než původní graf. Maximální počet kroků nutných k vyřešení multi-agentního plánování na grafu s n vrcholy je ovšem shora omezený $\mathcal{O}(n^3)$ kroky [14]. Maximálně tedy musíme budovat graf s $\mathcal{O}(n^3)$ vrstvami o n vrcholech. Výsledný časově expandovaný graf, na kterém hledáme multi-komoditní tok, je tedy veliký maximálně $\mathcal{O}(n^4)$.

5.2 Odvození heuristiky

V této podkapitole představíme námi navrženou heuristiku pro algoritmus A*. Vrátíme se tedy zpátky k multi-agentnímu plánování, tak jak jsme jej představili na začátku, tedy bez rotací, a budeme jej řešit pomocí popsaného algoritmu A*. K němu zkonstruujeme heuristiku odhadující makespan relaxací předchozího řešení problému. Relaxace bude spočívat v hledání maximálního toku namísto multi-komoditního toku. Tím anonymizujeme cíle agentů, tedy budeme hledat cestu agentů do libovolných cílů tak, aby každý došel do různého cíle. Hodnota

heuristiky je počet přechodů mezi vrstvami časově expandovaného grafu, tedy počet vrstev - 1. Nejprve si opět definujeme tok.

Definice 10. *Mějme orientovaný graf* $G = (V,E)$*, kde každá hrana* $(u,v) \in E$ *má přiřazenu celočíselnou kapacitu* $cap(u,v)$*. Navíc máme v grafu vrcholy* $s,t \in V$*, které označujeme jako zdroj a stok. Této čtveřici* (G,cap,s,t) *říkáme* síť. Tok *je ohodnocení* $f : E \to \mathbb{R}$*, které musí splňovat následující podmínky:*

1. *Podmínky kapacity*
$$0 \leq f(u,v) \leq cap(u,v)$$
2. *Kirchhoffův zákon*
$$\sum_{w \in V} f_i(u,w) = 0$$
pro $u \neq s,t$

Úkol nalezení maximálního toku je poté nalezení maximálního možného ohodnocení pro danou síť. Pro každou síť existuje nějaký maximální tok. Navíc pokud jsou všechny kapacity celočíselné, pak je tento tok také celočíselný [5].

Použití toku jako heuristiky nám přidá navíc interakci mezi agenty na rozdíl od heuristiky nejdelší z nejkratších cest. Pseudoalgoritmus pro naši heuristiku můžeme vidět na Algoritmu 2.

Algorithm 2 Heuristika pomocí toku

```
1:  procedure FlowHeuristic (G, A, α_k, α_+)
2:      expandedG ← ∅
3:      flow = 0
4:      i = getMaximalShortestPath(G, α_i, α_+)
5:      while tok < |A| do
6:          i = i + 1
7:          expandedG ← buildExpandedGraph(G, i)
8:          addSourceAndSink(expandedG, α_i, α_+)
9:          flow = maxFlow(expandedG)
10:     return i − 1
```

Heuristika potřebuje znát graf, ze kterého se bude stavět časově expandovaný graf, množinu agentů, respektive pouze její velikost, stav, pro který hledáme heuristický odhad a cílový stav. Na začátku inicializujeme expandovaný graf na prázdný, vynulujeme největší nalezený tok a nastavíme iniciální počet vrstev podle funkce `getMaximalShortestPath`(řádky 2-4). Tato funkce spočítá nejdelší z nejkratších cest agentů do jejich cíle stejně jako dříve zmíněná heuristika pro makespan. Výpočet nejkratších cest a nalezení maxima z nich je výpočetně méně náročné než hledání maximálního toku. Navíc víme, že nejdelší z nejkratších cest je spodní odhad, je tedy zbytečné toto znova počítat náročnější metodou. My chceme nalézt pomocí toku těsnější spodní odhad.

Požadovaná velikost hledaného toku musí být alespoň počet agentů (řádek 5). Když máme expandovaný graf s jednotkovou kapacitou všech hran, pak velikost toku odpovídá počtu agentů, kteří našli cestu ze své pozice do nějakého z cílů. Jak bylo již zmíněno, krokům odpovídají přechody mezi vrstvami, proto

zvětšíme i na začátku cyklu, aby toto pravidlo platilo i v první iteraci (řádek 6). Dále přímo postavíme expandovaný graf, tak jak jsme jej definovali, a navíc přidáme zdroj a stok (řádky 7 a 8). Zdroj a stok budou nové vrcholy. Zdroji přidáme hrany, které vedou do vrcholů v 0-té vrstvě, které odpovídají vrcholům, kde jsou umístěni agenti ve stavu α_k. Do stoku pak vedou hrany z poslední vrstvy z vrcholů, které odpovídají koncovému stavu α_+ (viz Obrázek 5.3). Všechny tyto nové hrany dostanou také jednotkovou velikost. To nám zaručí, že pro každého agenta, kterému jsme našli cestu z jeho počátečního vrcholu do nějakého cíle, jsme našli právě jednu cestu. Ukončovací podmínka tak skutečně platí, pouze pokud máme nějakou cestu pro každého agenta. Dále na takto vybudovaném grafu nalezneme maximální tok (řádek 9). Návratová hodnota je $i - 1$, což odpovídá počtu přechodů mezi vrstvami.

Tvrzení 8. *Mějme zadání multi-agentního plánování s rotacemi* (G,A,α_0,α_+) *pro čistě orientovaný graf* G*. Navíc nechť má toto zadání řešení v makespanu* T*. Pak maximální tok nad časově expandovaným grafem* G_{T+1} *s* $T+1$ *vrstvami odpovídá cestám anonymizovaných agentů.*

Důkaz. Maximální tok na grafu G_{T+1} má velikost maximálně $|A|$, protože zdroj i stok mají výstupní, respektive vstupní, stupeň $|A|$ a všechny hrany v grafu mají jednotkovou kapacitu. Každý vrchol, mimo zdroje a stoku, má buďto vstupní nebo výstupní stupeň rovný jedné. V síti navíc existuje celočíselný maximální tok. Díky tomu nám tok udává až $|A|$ cest ze zdroje do stoku, které jsou vrcholově disjunktní. To že těchto cest musí být právě $|A|$ a že odpovídají právě možným pohybům agentů, dostáváme z definice časově expandovaného grafu a Pozorování 1.

□

Když se podíváme na příklad z Obrázku 5.3, můžeme si všimnout, že nalezený tok neodpovídá skutečným cestám agentů, ale cestám anonymizovaných agentů podle Tvrzení 8. Podle tohoto řešení by zelený agent došel do cíle modrého a naopak. V heuristice nás ovšem konkrétní cesty nezajímají, pouze počet kroků. Už v tomto jednoduchém příkladu nám dává naše heuristika lepší odhad než heuristika nejdelší z nejkratších cest. Ta by v tomto příkladu byla rovna třem, zatímco naše heuristika je rovna čtyřem. Vždy platí, že heuristika pomocí toku dává alespoň tak dobrý odhad, jako heuristika nejdelší z nejkratších cest podle následujícího pozorování.

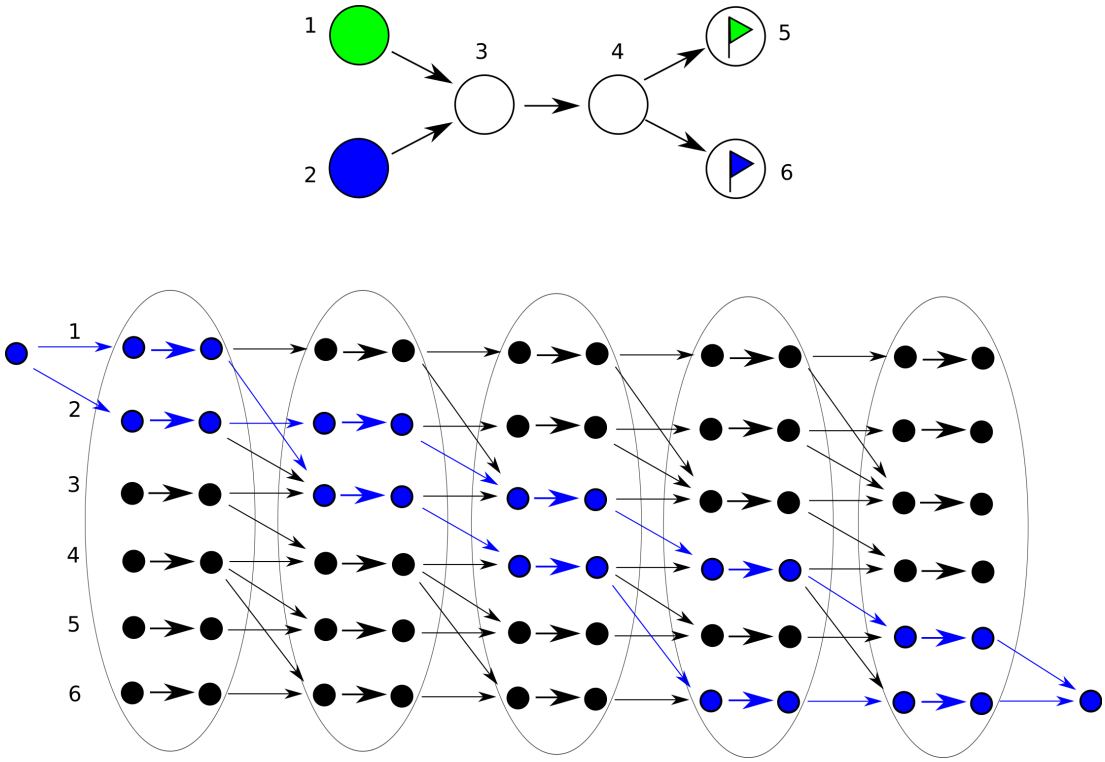
Pozorování 9. *Pro každý stav* α_i *je hodnota heuristiky pomocí toku větší nebo rovna hodnotě heuristiky nejdelší z nejkratších cest.*

Důkaz. Jednoduše vyplývá z počátečního nastavení počtu vrstev časově expandovaného grafu. I kdyby existoval tok požadované velikosti na grafu s menším počtem vrstev, heuristika vrátí toto dané minimum.

□

Navíc ve složitějších příkladech s více agenty může toková heuristika dříve rozeznat, že se nějaký agent pohnul špatným směrem. Pokud se agent, který nemá do cíle nejdelší cestu pohne špatným směrem, pak to heuristika nejdelší z nejkratších cest ignoruje.

Obecně nemusí platit, že dostaneme přesný odhad makespanu, vždy ale dostaneme spodní odhad nutného počtu kroků, to vyplývá z konzistence heuristiky. Tato vlastnost je také důležitá pro použití v naší verzi A*.



Obrázek 5.3: **Časově expandovaný graf pro heuristiku pomocí toku.** Nahoře je vidět původní graf s agenty a vlajkami, které reprezentují jejich cíle. Ve spodní části je časově expandovaný graf pro tento stav s přidaným zdrojem a stokem a nalezeným maximálním tokem.

Tvrzení 10. *Heuristika pomocí toku je konzistentní.*

Důkaz. Podobně jako v pozorováních 4 a 6 chceme ukázat, že platí vztah

$$h(n) \leq c(n,k) + h(k),$$

pro všechny stavy n a jeho potomky k. V jednom časovém okamžiku provedou krok všichni agenti (platným krokem je i stání na místě) a tím se dostaneme do nového stavu. Platí tedy $c(n,k) = 1$. Předpokládejme nyní, že $h(n)$ i $h(k)$ nejsou rovny odpovídající hodnotě heuristiky nejdelší z nejkratších cest. V takovém případě by tvrzení platilo podle pozorování 6. Tyto hodnoty jsou navíc spodní odhad pro heuristiku pomocí toku.

Nechť $|A|$ je počet agentů a $h(n) = j - 1$. Potom j musí být nejmenší číslo takové, že v časově expandovaném grafu G_j s j vrstvami nalezneme tok velikosti $|A|$. Tento tok označme f_j. Nyní využijeme tvrzení 1 o pozicích agentů v původním grafu a v časově expandovaném grafu. Stav n odpovídá rozmístění agentů v 0-té vrstvě grafu G_j. Stav k získáme přesunutím všech agentů do 1. vrstvy po nějaké hraně. Nyní hledáme minimální počet vrstev, aby v časově expandovaném grafu existoval tok velikosti $|A|$, za předpokladu, že zdroj je napojený na vrcholy

odpovídající stavu k. Pokud se všichni agenti přesunuli po hranách, které mohly být nasycené v toku f_j, pak je toto číslo $j-1$, jinak by to byl spor s minimalitou j. Na druhou stranu, pokud se někteří agenti přesunuli po jiných hranách, musí toto číslo být alespoň j, jinak by to byl opět spor s minimalitou j.

□

Díky tomuto tvrzení můžeme naši heuristiku využívat efektivně v algoritmu A* bez nutnosti opětovně navštěvovat již jednou uzavřené vrcholy.

Jedno z možných vylepšení představené heuristiky, co se doby výpočtu týká, je chytré stavění časově expandovaného grafu. V pseudoalgoritmu neřešíme, jakým způsobem využít již postavenou část grafu, když se přesuneme do další iterace. Vzhledem k vrstevnaté struktuře časově expandovaného grafu je jednoduché využít již existující vrstvy a pouze přidat jednu novou. Následně pak stačí jen upravit stok a hrany pro něj přidané. Obdobným způsobem můžeme využít dříve zkonstruovaný graf při jiném volání samotné heuristiky. V takovém případě máme již zkonstruovaný graf s nějakým počtem vrstev n. Pokud tento počet vrstev není dostačující pro současné volání, přidáme další. Pokud naopak má graf více vrstev, než je potřeba, přidáme pouze stok k žádoucí vrstvě k. Vrstvy $k+1, \ldots, n$ pak nebudou využity při hledání toku, jelikož nevede hrana do nižší vrstvy.

Poslední poznámkou k využití navržené heuristiky je, pro jaký druh multi-agentních pohybů ji využíváme. A* tak, jak jsme jej popsali, nedovoluje pohyby agentů po obsazených orientovaných kružnicích. Jeho modifikace na dovolení těchto pohybů by byla náročná, protože by bylo potřeba najít všechny kružnice v grafu a v každém kroku kontrolovat, zda není možné po některé z nich tento pohyb provést. Na druhou stranu naše heuristika pomocí toku ale agentům pohyb po kružnicích nezakazuje. Je totiž relaxací řešení multi-agentního plánování pomocí multi-komoditního toku, kde jsou tyto pohyby povolené. V obecném případě je ale větší limitací heuristiky anonymizování cílů než předpoklad, že se jedná o multi-agentní plánování s rotacemi.

5.3 Heuristika v neorientovaných grafech

Při popisování námi navržené heuristiky jsme přímo nekladli požadavky na vstupní graf. Konkrétně zda se má jednat o čistě orientovaný nebo neorientovaný graf. Podle konstrukce časově expandovaného grafu by se mohlo zdát, že heuristika bude na neorientovaných grafech fungovat hůře a je pro ně potřeba stavět časově expandované grafy za pomocí grafové konstrukce popsané u multi-komoditního řešení problému (viz Obrázek 5.2). V případě převodu na hledání nejlevnějšího multi-komoditního toku v síti jsme vyžadovali, aby se jednalo o čistě orientovaný graf, jinak by nalezené cesty nebyly korektní. Nyní ovšem ukážeme, že i pro neorientované grafy bude heuristika dávat stejné hodnoty při použití námi definované konstrukce časově expandovaného grafu. Tato konstrukce na rozdíl od použití přidaných vrcholů dovoluje prohození agentů (viz Obrázek 5.4).

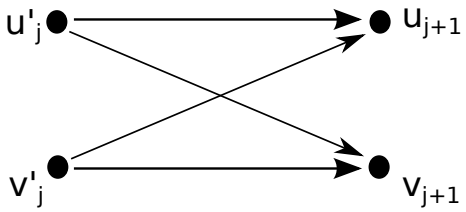
Tvrzení 11. *Povolení prohození agentů v časově expandovaném grafu při výpočtu heuristiky pomocí toku dává stejnou hodnotu jako při jeho zakázání.*

Důkaz. Při výpočtu heuristiky pomocí toku jsou anonymizované cíle, tedy i agenti jsou anonymní. Prohození dvou agentů nám tedy nezmění stav a je ekvivalentní

situaci, kdy se oba tito agenti nepohnuli. Pokud tedy nalezneme minimální počet vrstev časově expandovaného grafu pomocí toku, kde došlo k prohození dvou agentů, mohli jsme stejného výsledku dosáhnout tokem, kde se tito dva agenti v daném časovém kroku nehýbají.

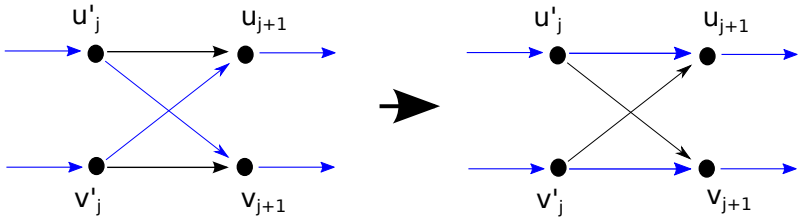
□

V některých předchozích tvrzeních jsme předpokládali, že se jedná o čistě orientované grafy. V těchto tvrzeních jsme ale spojovali nalezený tok s naplánovanými cestami agentů. Od heuristiky ale očekáváme pouze numerický odhad makespanu do cílového stavu. Jak jsme ukázali v Tvrzení 11, je tato hodnota stejná. Naši heuristiku můžeme tedy beze změny a s lehčí konstrukcí časově expandovaného grafu používat i v neorientovaných grafech.



Obrázek 5.4: **Expandovaný graf pro neorientovanou hranu.** Pokud mezi vrcholy u a v vede neorientovaná hrana, pak takto vypadá přechod mezi dvěma vrstvami časově expandovaného grafu.

Na základě tohoto pozorování můžeme doporučit možné vylepšení k existujícímu algoritmu, který řeší multi-agentní plánování, kde jsou agenti rozděleni do skupin netriviálním způsobem. Tedy se nejedná o jednu skupinu obsahující všechny agenty ani o počet skupin rovný počtu agentů. Tyto varianty jsme popisovali dříve v Kapitole 3. Tento algoritmus ConflictBased Min-Cost-Flow (CBM) [10] je založený na prohledávacím algoritmu CBS, který jsme již také zmiňovali.



Obrázek 5.5: **Úprava toku v časově expandovaném grafu.** Barevné hrany značí nasycené hrany. Tato úprava zajistí, že agenti místo prohození stojí na místě.

Na vyšší úrovni opět staví a prohledává strom obsahující podmínky pro zamezení nalezených konfliktů. Na nižší úrovni ovšem plánuje cestu nikoliv pro jednotlivé agenty, ale pro jednotlivé skupiny agentů. Cesty pro jednotlivé týmy agentů se hledají pomocí nalezení maximálního toku na expandovaném grafu. Autoři v článku ovšem využívají grafové konstrukce z Obrázku 5.2. Tento nápad přímo vychází z plánování cest pomocí multi-komoditního toku. Jak jsme nyní ukázali, pro validaci cesty anonymních agentů nemusíme tuto konstrukci využívat. Tok je tak možné hledat na grafu, který je až o $(k-1)2|E|$ vrcholů menší

a $(k-1)3|E|$ hran menší, kde k je počet vrstev časově expandovaného grafu a $|E|$ je počet hran původního grafu. Tato čísla vyplývají jednoduše z porovnání Obrázků 5.2 a 5.4. Za každou hranu si ušetříme 2 vrcholy a 3 hrany a toto děláme pro $k-1$ přechodů mezi k vrstvami. Tyto nalezené cesty nebudou odpovídat skutečným cestám agentů. Pro validaci, zda je plán bez konfliktů, ovšem stačí. Pokud nalezneme cílový vrchol, pak tyto cesty můžeme jednoduše upravit na skutečné cesty jedním průchodem do hloubky časově expandovaného grafu. Pokud existuje nasycená hrana $\langle u'_j,v_{j+1}\rangle$ a zároveň je nasycena hrana $\langle v'_j,u_{j+1}\rangle$, tak došlo k prohození agentů. Cesty jednoduše upravíme tak, že těmto hranám vynulujeme tok a nasytíme hrany $\langle u'_j,u_{j+1}\rangle$ a $\langle v'_j,v_{j+1}\rangle$, které jistě nasycené nebyly (viz Obrázek 5.5). Zda je toto vylepšení v praxi znatelné, v této práci nebudeme zkoumat. Jedná se pouze o postřeh vyvozený z našeho výzkumu.

6. Výsledky experimentů

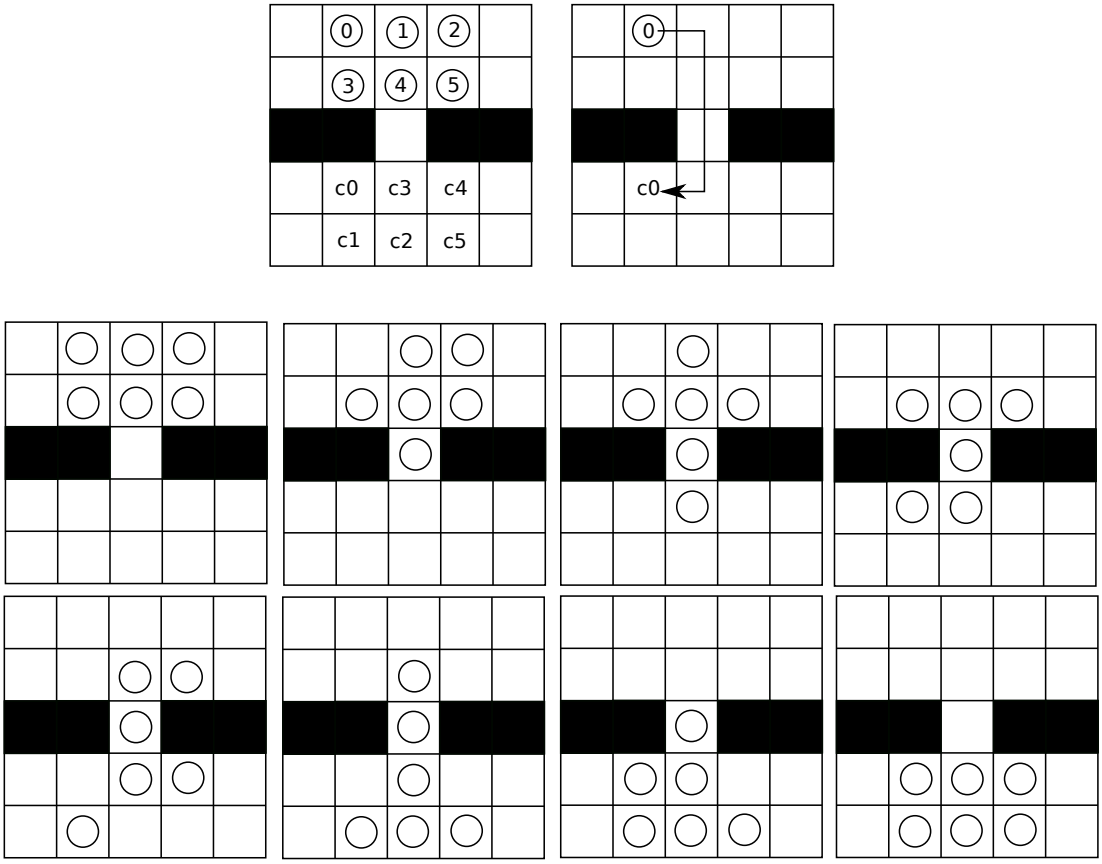
K otestování nově navržené heuristiky provedeme sérii experimentů. V těchto experimentech budeme využívat námi napsané implementace algoritmu A* tak, jak byl popsán v dřívější kapitole. Tato implementace používá funkci g_m, tedy minimalizujeme makespan. Jako heuristiky, tedy funkce h, používá podle parametrů programu buďto heuristiku pomocí toku, nebo heuristiku nejdelší z nejkratších cest. Po zbytek této práce budeme heuristiku pomocí toku označovat jako $flowheuristic$, $flow$ nebo h_f. Obdobně heuristiku nejdelší z nejkratších cest budeme označovat jako $shortheuristic$, $short$ nebo h_s. Implementace základního algoritmu je stejná pro obě použité heuristiky, rozdíl ve výkonu je tedy pouze na základě použitých heuristik. Detailní informace k implementaci algoritmu A* se nacházejí v příloze Implementační detaily.

Všechny testy probíhaly na stejném počítači. Tímto počítačem je virtuální stroj vytvořený pomocí programu VMware Workstation verze 12.0.0. Na virtuálním stroji je operační systém Ubuntu verze 14.04 64-bit. Tento virtuální stroj využívá dvě jádra procesoru a dva gigabyte fyzické paměti fyzického počítače. Parametry fyzického počítače jsou následující: Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz, Windows 10 64-bit. Program je napsaný v jazyce C++ a přeložený pomocí překladače gcc verze 4.8.4.

6.1 Testovací instance

Jak bylo zmíněno dříve, heuristika flow bere v úvahu interakci mezi agenty při přesouvání. Zabraňujeme totiž více agentům být po cestě v jednom vrcholu, zatímco heuristika short bere v úvahu pouze nejvzdálenějšího agenta a neřeší, jak se bude po cestě potkávat s dalšími agenty (viz Obrázek 6.1, který uvádí konkrétní příklad z testovacích instancí). Navíc pokud se nějaký agent, který nemá před sebou nejdelší cestu ze všech, pohne špatným směrem, heuristika short na to nereaguje včas, zatímco heuristika flow má tendenci posouvat stále všechny agenty směrem k cíli. Na druhou stranu heuristika flow nerozeznává jednotlivé cíle, dává tedy lepší odhad, pokud jsou cíle blízko sebe, a konečné uspořádání agentů v rámci cílů se najde hrubou silou. Zadání, kde jsou agenti i jejich cíle rozmístěny náhodně, tedy nedostávají lepší odhad, protože agenti jsou naváděni do cílů tak, aby byl makespan co nejmenší, což většinou nebude správný cíl. Ovšem díky tomu, že tok začínáme hledat na počtu vrstev, který odpovídá nejdelší z nejkratších cest, tak nedostaneme nikdy horší odhad než u heuristiky short. V takovýchto náhodných grafech ale trvá výpočet značně déle, protože zbytečně stavíme expandovaný graf a hledáme tok, který nedá lepší odhad.

Z těchto důvodů předpokládáme, že heuristika flow bude úspěšnější v zadání, kdy se pohromadě přesouvá skupina agentů z jednoho místa do jiného, a navíc je nutné obejít nějakou překážku, čímž se vynutí jejich interakce. Pro testování jsme vytvořili sadu náhodně generovaných zadání. Tato zadání jsou vytvořena na dvou typech mřížkových grafů. Prvním je mřížka, která obsahuje úzké hrdlo (viz Obrázek 6.2 vlevo). Tento typ grafu budeme dále značit jako $BottleNeck$ nebo BN. Druhým je mřížka, která uprostřed obsahuje překážku, kterou je nutno obejít (viz Obrázek 6.2 vpravo). Tento typ grafu budeme dále značit jako $Obstacle$ nebo



Obrázek 6.1: **Vizualizace naplánovaných cest pomocí jednotlivých heuristik.** Nahoře je vidět zadání a heuristika short, kde se uvažuje pouze cesta jednoho agenta bez interakce se zbytkem skupiny. Dole je vidět naplánovaný pohyb všech agentů pomocí heuristiky flow. Agenti jsou záměrně neoznačeni, z důvodu anonymizace agentů při použití heuristiky. V tomto případě je řešení problému v makespanu 7, což je odhad heuristiky flow. Odhad short je 5.

OB. Obě tyto mřížky mají hranu velikosti 5, tedy celkově obsahují 25 vrcholů.

Na těchto grafech vytvoříme několik typů zadání, která se budou lišit ve startovních a cílových pozicích agentů. Pro oboje máme dvě možnosti - pohromadě a náhodně rozmístěné. Použijeme tak následující tři typy zadání

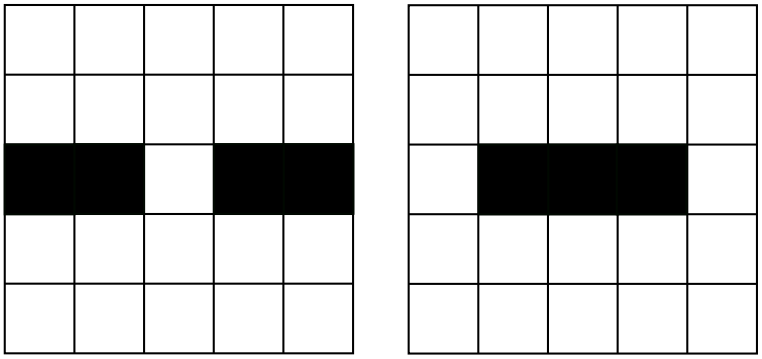
1. Start pohromadě, cíl pohromadě.
   Budeme označovat $centralized - centralized$ $(c - c)$.

2. Start náhodně, cíl pohromadě.
   Budeme označovat $scattered - centralized$ $(s - c)$.

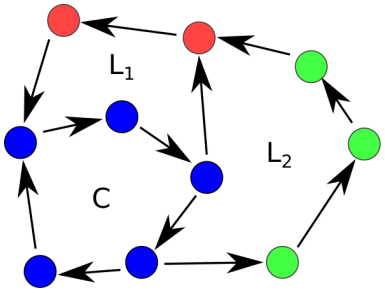
3. Start náhodně, cíl náhodně.
   Budeme označovat $scattered - scattered$ $(s - s)$.

Čtvrtou možnost, kterou je start pohromadě a cíl náhodně, nebudeme uvažovat, protože při použitých heuristikách je obdobný třetí uvedené možnosti.

Poslední zcela odlišné zadání bude čistě orientovaný 2-souvislý graf vytvořený jako kružnice, ke které jsou přidané uši[19] (viz Obrázek 6.3). Kružnice obsahuje 5 vrcholů a každé ze 4 přidaných uší obsahuje 5 vrcholů. Celková veliksot grafu



Obrázek 6.2: **Mřížkové grafy použité při testování heuristik.** Černá pole představují překážky.



Obrázek 6.3: **Příklad čistě orientovaného 2-souvislého grafu.** Na obrázku je vidět rozložení na kružnici a uši. V testování heuristik použijeme větší instance grafu.

je tedy také 25 vrcholů. Rozmístění agentů v tomto grafu bude zcela náhodné, stejně tak jako jejich cílů. Tento typ grafu budeme značit jako *Oriented* nebo *OR*. Všechna možná zadání jsou uvedena v Tabulce 6.1. Generátory i vstupní a výstupní formát programu jsou detailně popsány v přílohách Vstupní formát, Výstupní formát a Generátory.

Typ grafu	Rozmístění agentů	#agentů	#instancí	Celkem instancí
OB	c-c	2-8	10	70
OB	s-c	2-8	10	70
OB	s-s	2-5	10	40
BN	c-c	2-8	10	70
BN	s-c	2-8	10	70
BN	s-s	2-5	10	40
OR	random	2-9	10	80

Tabulka 6.1: **Instance použité pro testování.**

Tyto parametry byly zvoleny tak, aby byly pozorovatelné rozdíly mezi použitými heuristikami, ale zároveň byl únosný výpočetní čas. Pro instance s málo agenty je rozdíl mezi heuristikami menší, ale výpočet je rychlejší. Na druhou

stranu pro více agentů je rozdíl mezi heuristikami znatelný, ale výpočetní čas se značně zvyšuje. Testování je proto náročné na velkém množství instancí. Konkrétní naměřené výsledky představíme v následující podkapitole.

6.2 Porovnání heuristik

V této podkapitole porovnáme algoritmus A* za použití heuristik short a flow. Při výpočtu dříve popsaných zadání problému měříme dvě veličiny - počet navštívených stavů a výpočetní čas. Počet navštívených stavů odpovídá počtu stavů ze stavěného stavového stromu, které jsou během výpočtu prozkoumány a uzavřeny. Toto číslo tedy odpovídá počtu stavů, které jsou na konci výpočtu označeny jako closed. Výpočetní čas přímočaře udává počet vteřin od začátku do konce výpočtu. Pro každou instanci problému jsme stanovili jako timeout 100 minut, tedy 6000 vteřin. V následujících grafech počítáme pouze ty instance, které byly vyřešeny ve stanoveném čase. Všechny grafy v následujících sekcích jsou setříděné instance podle dané měřené veličiny (čas nebo počet stavů). X-ová osa tedy odpovídá pořadí instance, zatímco y-ová osa odpovídá času, respektive počtu navštívených stavů, který uplynul, respektive bylo navštíveno, během výpočtu. V každém grafu jsou pak dvě křivky, jedna pro každou heuristiku. Pokud některá z nich neobsahuje všechny hodnoty, pak to znamená, že tato instance nebyla vyřešena v časovém limitu.

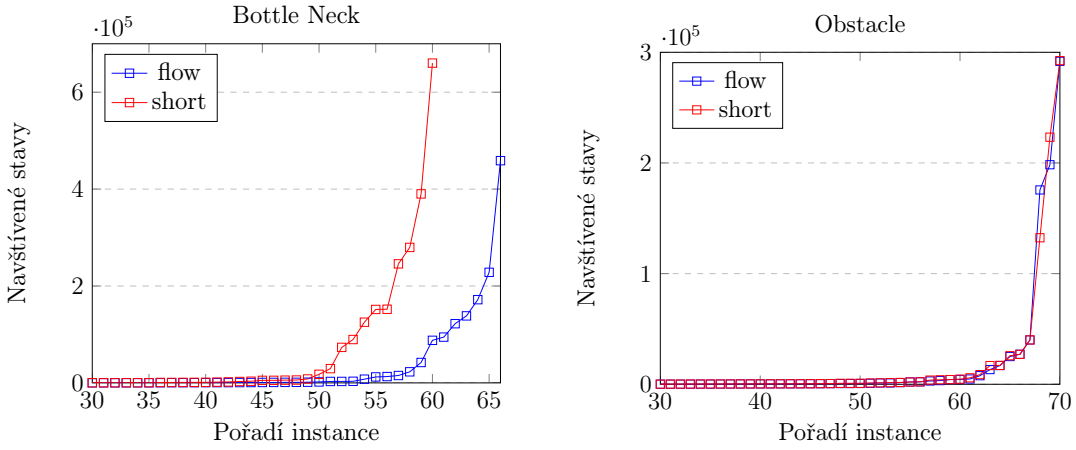
Vzhledem k tomuto setřídění a růstu složitosti instancí na grafech nezobrazujeme prvních několik instancí, které jsou vyřešeny nejrychleji nebo za navštívení nejméně stavů. Na těchto instancích není patrný žádný rozdíl mezi heuristikami ani není vidět růst složitosti. Navíc zúžením grafu budou lépe pozorovatelné rozdíly u složitých instancí. Grafy jsou rovnány tak, že vlevo je graf odpovídající typu Bottle Neck a vpravo typu Obstacle.

V následujících čtyřech sekcích prozkoumáme výsledky pro různé počáteční a cílové rozmístění agentů na mřížkových grafech a pro čistě orientované grafy.

6.2.1 centralized - centralized

Jako první porovnáme rozmístění centralized - centralized. Na grafech na Obrázku 6.4 můžeme vidět počet navštívených stavů pro oba typy grafů. Můžeme si všimnout, že pro grafy typu Bottle Neck roste počet navštívených stavů u heuristiky short mnohem razantněji než pro heuristiku flow. Navíc heuristika flow vyřešila o 6 více instancí v časovém limitu. Konkrétně short nevyřešila žádnou z instancí s 8 agenty a flow vyřešila 6 z 10 těchto instancí.

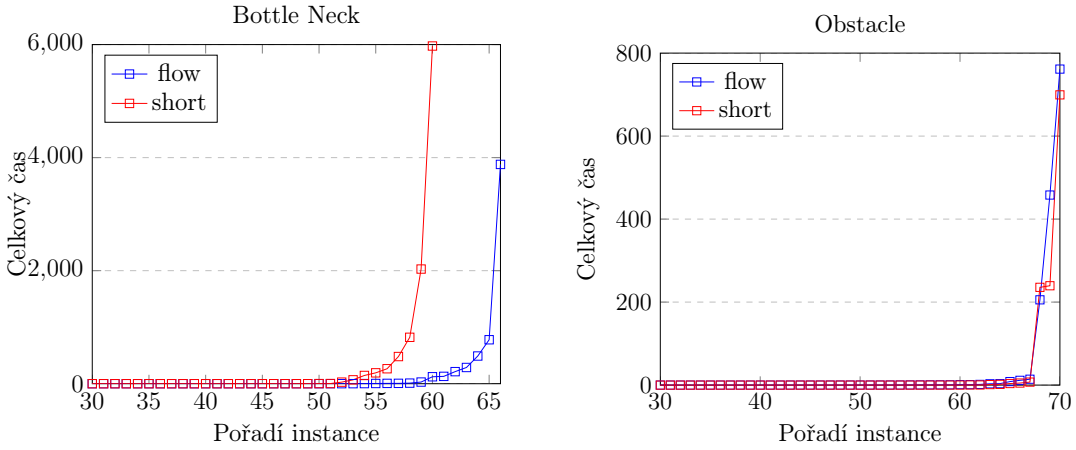
Pro grafy typu Obstacle není rozdíl v počtu navšívených stavů tolik znatelný. V některých instancích je pak úspěšnější short a v jiných naopak flow. Tento rozdíl však neni nijak veliký.



Obrázek 6.4: **Počet navštívených stavů c-c.** Vlevo je vidět počet navštívených stavů pro grafy typu BN. Vpravo je vidět počet navštívených stavů pro grafy typu OB.

Pokud porovnáme celkový výpočetní čas pro oba typy grafů (viz Obrázek 6.5), vidíme, že heuristika flow je úspěšnější na grafech Bottle Neck. To odpovídá tomu, že v těchto instancích prozkoumává i méně stavů. Konkrétně pro instance se 7 agenty byla flow více než desetkrát rychlejší než short. Instance s nejdelším výpočetním časem pro flow je instance s 8 agenty, zatímco pro short má nejdéle počítaná instance agentů 7 (short nevyřešila žádnou instanci s 8 agenty). Nejdelší instance řešená pomocí flow je tak vyřešena rychleji než nejdelší instance řešená short, i když jsou s různým počtem agentů.

Na druhou stranu na grafech typu Obstacle byla obecně úspěšnější heuristika short. To opět odpovídá počtu navštívených stavů. Pokud obě heurisitiky prozkoumávají podobný počet stavů je výpočetní čas delší pro flow, jelikož je náročnější na výpočet. Znatelný rozdíl je ale až u posledních 3 instancí. Pro tento typ grafu byla včas vyřešena všechna zadání.

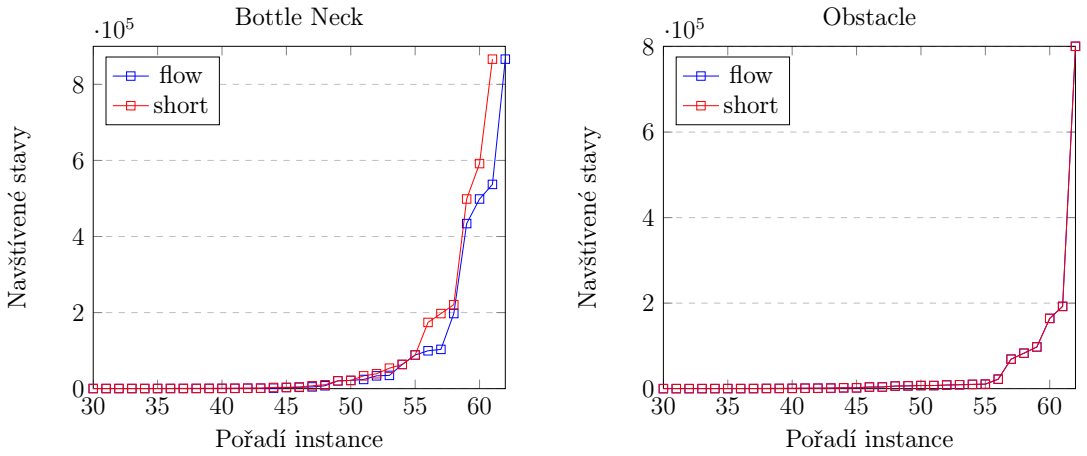


Obrázek 6.5: **Celkový čas c-c.** Vlevo je vidět celkový čas pro grafy typu BN. Vpravo je vidět celkový čas pro grafy typu OB.

6.2.2 scattered - centralized

Dalším rozmístěním je scattered - centralized. Opět můžeme vidět na Obrázku 6.6 počty navštívených stavů. I v tomto případě je v grafech Bottle Neck úspěšnější heuristika flow. Rozdíl ale není tolik znatelný a začíná být patrný až u složitějších instancí s více agenty. To je způsobeno náhodným rozmístěním počátečních pozic agentů. V případě, že se v prostředí vyskytuje malý počet agentů, je pravděpodobnost jejich interakce menší, než když je graf zaplněný. Při tomto typu rozmístění byly obě heuristiky méně úspěšné, co se týče počtu vyřešených instancí v časovém limitu (viz Tabulka 6.2). Flow celkově vyřešila o jednu instanci více než short.

Počet navštívených stavů pro grafy typu Obstacle je takřka totožný pro obě heuristiky ve všech instancích. Rozdíly jsou maximálně v desítkách stavů, což na grafu není pozorovatlené. Jedinou poznámkou je, že celkový počet navštívených stavů je razantně menší než u zadání stejné velikosti nad grafy Bottle Neck s vyjímkou jedné instance.



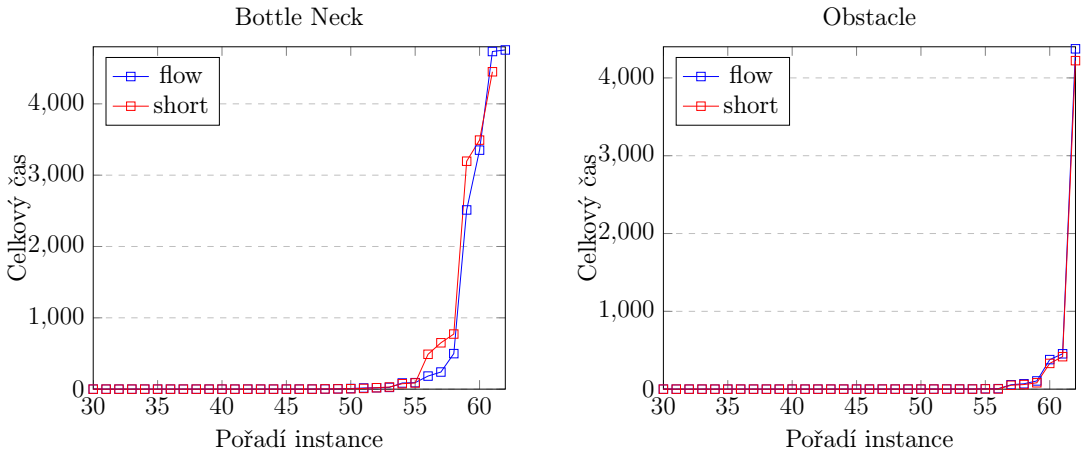
Obrázek 6.6: **Počet navštívených stavů s-c.** Vlevo je vidět počet navštívených stavů pro grafy typu BN. Vpravo je vidět počet navštívených stavů pro grafy typu OB.

	2-6 agentů		7 agentů		8 agentů	
	BN	OB	BN	OB	BN	OB
flow	100%	100%	80%	70%	40%	50%
short	100%	100%	80%	70%	30%	50%

Tabulka 6.2: **Úspěšnost instancí s-c.**

Celkový výpočetní čas (viz Obrázek 6.7) odpovídá naměřenému počtu navštívených stavů. Pro grafy Bottle Neck je rychlejší flow, i když zobrazený rozdíl se nezdá být příliš velký. Ve většině složitých instancí, s jedinou vyjímkou, je rychlejší flow. Ta navíc vyřešila jednu instaci, kterou short nezvládla v časovém limitu.

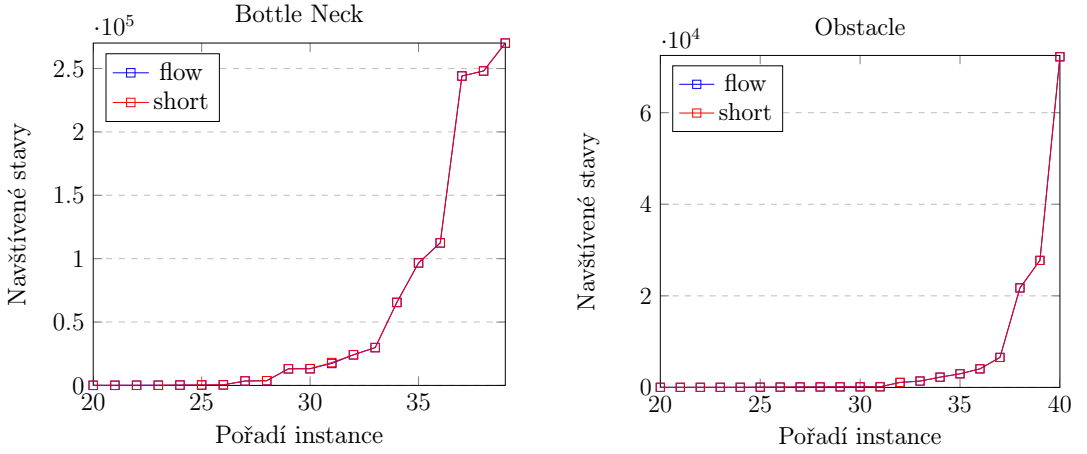
Pro grafy typu Obstacle také platí, že rozdíly v času nejsou příliš zaranzní. V tomto případě je ovšem rychlejší heuristikou short. Opět to plyne z toho, že počet navštívených stavů je podobný, ale potřebný čas pro vypočtení heuristiky flow je delší.



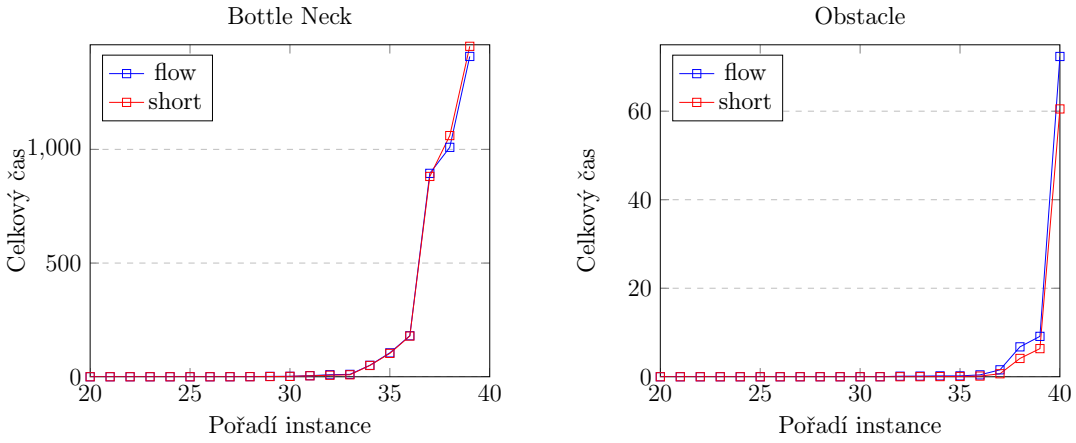
Obrázek 6.7: **Celkový čas s-c.** Vlevo je vidět celkový čas pro grafy typu BN. Vpravo je vidět celkový čas pro grafy typu OB.

6.2.3 scattered - scattered

Pro úplnost uvádíme i poslední uvažované rozdělení agentů a to scattered - scattered. V tomto případě obě heuristiky na obou typech grafů prozkoumají prakticky stejné množství stavů a rozdíly jsou maximálně v desítkách a tedy nejsou viditelné (viz Obrázek 6.8). Na výpočetním čase tyto rozdíly také nejsou patrné (viz Obrázek 6.9), porotože jsme se v tomto rozmístění omezili pouze na malé instance. Ve větších instancích by rozdíl začal znatelně růst. Ty jsme ale nezahrnuli, protože v ostatních experimentech jsme ukázali, že při stejném množství navštívených stavů roste výpočetní čas heuristiky flow rychleji než heuristiky short.



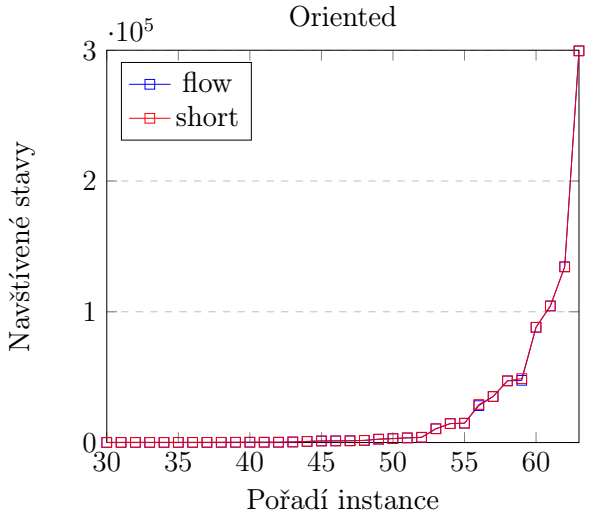
Obrázek 6.8: **Počet navštívených stavů s-s.** Vlevo je vidět počet navštívených stavů pro grafy typu BN. Vpravo je vidět počet navštívených stavů pro grafy typu OB.



Obrázek 6.9: **Celkový čas s-s.** Vlevo je vidět celkový čas pro grafy typu BN. Vpravo je vidět celkový čas pro grafy typu OB.

6.2.4 Oriented

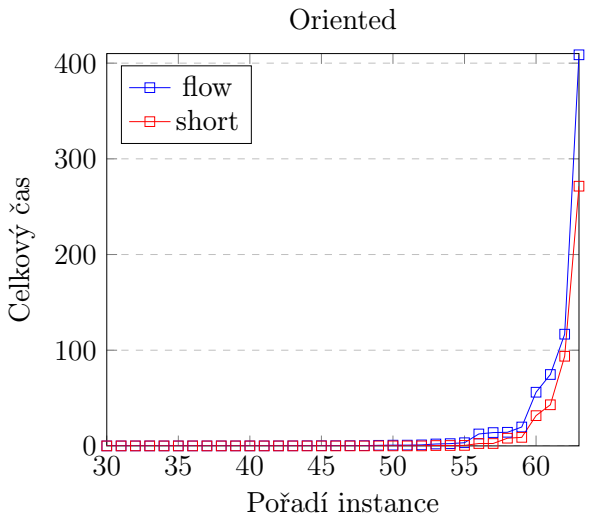
Jako experiment pro porovnání výpočetního času volání obou heuristik jsme vytvořili náhodné řešitelné instance. Pro zajištění řešitelnosti jsme využili čistě orientované 2-souvisé grafy, na kterých je instance multi-agentního plánování vždy řešitelná, pokud existují alespoň dva volné vrcholy [1]. Do těchto instancí jsme rozmístili náhodně startovací i cílové pozice agentů. Náš předpoklad, že obě heuristiky budou prozkoumávat stejný počet stavů, se potvrdil (viz Obrázek 6.10). Všechny vyřešené instance, s vyjímkou dvou, vyžadovaly prozkoumání stejného počtu stavů pro obě heuristiky.



Obrázek 6.10: **Počet navštívených stavů oriented.**

Jak můžeme vidět na Obrázku 6.11, náš předpoklad se potvrdil a volání heuristiky flow je časově náročnější. V nejsložitějších příkladech je rozdíl téměř dvakrát tak velký. Tento rozdíl jsme očekávali z toho důvodu, že v heuristice flow provádíme nejprve celý výpočet, který probíhá v heuristice short a navíc stavíme časově expandovaný graf, na kterém hledáme maximální tok. V případech, kdy

tento extra výpočet nepřinese těsnější spodní odhad a tím navštívení méně stavů, nemůžeme dosáhnout lepšího celkového výpočetního času.



Obrázek 6.11: **Celkový čas oriented.**

6.3 Zhodnocení heuristik

Na dříve uvedených zadání jsme testovali námi nově navrženou heuristiku, která využívá toků v síti nad expandovaným grafem. Tato heuristika dává odhad makespanu z daného stavu do cílového stavu aproximací cest pro všechny agenty. Tyto cesty jsou bez kolizí, na druhou stranu aproximace spočívá v anonymizaci agentů a jejich cílů. Jako známou heuristiku, oproti které jsme testovali naši novou heuristiku, jsme zvolili heuristiku nejdelší z nejkratších cest. Tato udává odhad makespanu z daného stavu do cílového stavu jako nejdelší z nejkratších cest všech agentů do jejich konkrétních cílů. Tato heuristika tak neanonymizuje agenty, na druhou stranu ale nebere v úvahu jejich interakci po cestě do cílů.

Z těchto vlastností jsme odhadli typy grafů a rozmístění agentů, pro které bude naše heuristika úspěšná, a pro které naopak neúspěšná. Na naměřených výsledcích z předchozí kapitoly jsme viděli, že na grafech typu Bottle Neck, tedy grafech s úzkým hrdlem, kterým musejí agenti projít, byla naše heuristika značně úspěšnější, jak v počtu navštívených stavů, tak v celkovém výpočetním čase. Úspěšnější byla v případech, kdy se agenti přesouvali po skupině zkrz zmíněné úzké hrdlo. Pokud se agenti předouvali z náhodných míst do jedné oblasti, byla naše heuristika stále úspěšnější v obou veličinách, ovšem ne tak významně. V případě, že se agenti přesouvají nezávisle na sobě, tedy z i do náhodných míst, není naše heuristika nijak úspěšnější pro žádný typ grafů.

Pro grafy typu Obstacle, tedy grafy, kde je uprostřed grafu překážka, kterou lze obejít ze dvou stran, jsme také čekali, že bude naše heuristika úspěšná. Experiemnt ale tento předpokald neoptvrdil pro libovolné rozložení agentů. Na druhou stranu naše heuristika nebyla tolik pozadu. Navíc pro tyto instance se řešení jevilo být jednodužší než pro grafy typu Bottle Neck, protože většina zadání byla splněna v časovém limitu. Je tedy možné, že pro větší instance by se projevil větší rozdíl ve zkoumaných heuristikách.

Jako experiment pro výpočetní čas jednotlivých volání obou heuristik jsme vytvořili i instance s grafy, kde jsme správně předpokládali, že obě heuristiky prozkoumají podobný počet stavů. Z tohoto experimentu jsme vyvodili, že volání heurisitiky flow je opravdu časově náročnější než heuristiky short. Na druhou stranu časový rozdíl se projevil až u nejsložitějších instancí a i v takovém případě nebyl více než dvakrát větší. V závěrečné kapitole také rozebíráme možnosti zrychlení výpočtu naší nové heuristiky, které mohou tento rozdíl ve výpočetním čase nadále snížit.

7. Závěr

V této práci jsme prozkoumali plánování optimálních cest pro skupinu agentů. Toto multi-agentní plánování jsme řešili prohledáváním stavového prostoru pomocí algoritmu A* a zároveň pomocí převodu na multi-komoditní toky nad časově expandovaným grafem. Naším přínosem v této práci je vytvoření nové heuristiky pro algoritmus A*, kterou jsme odvodili právě ze zmíněného převodu na multi-komoditní toky. Heuristika hledá cesty pro agenty nad časově expandovaným grafem, kde každá vrstva odpovídá jednomu časovému okamžiku. Hledání cest probíhá pomocí hledání maximálního jedno-komoditního toku, který lze najít v polynomiálním čase. Touto relaxací ovšem anonymizujeme agenty, kteří se tak mohou dostat do cizích cílů. Z tohoto důvodu se jedná o heuristický odhad a nikoliv o přesné plánování cest. Navíc jsme také ukázali, že konstrukce časově expandovaného grafu pro čistě orientované grafy dává stejné hodnoty heuristiky jak pro orientované, tak pro neorientované grafy.

O této heuristice jsme ukázali, že je konzistentní a tedy i přípustná. Z toho vyplává, že jí lze využít efektivně v algoritmu A* bez nutnosti opětovně navštěvovat již jednou uzavřené vrcholy. Dále jsme teoreticky popsali její výhody v odhadu celkového makespanu do cílového stavu. Tyto výhody vyplývají z toho, že více bere v úvahu interakci mezi agenty při pohybu. Na druhou stranu jsme popsali i její nevýhody, které spočívají v anonymizaci agentů. Z těchto poznatků jsme teoreticky přepověděli typy zadání, ve kterých bude naše heuristika úspěšná.

Implementovali jsme naší verzi algoritmu A* a pro ni jsme zároveň implementovali nově navrhovanou heuristiku společně s dalšími dobře známými heuristikami. S těmito programy jsme provedli řadu experimentů pro porovnání heuristik. Při porovnávní jsme se zaměřili na dvě veličiny výpočtu - počet navštívených stavů a celkový výpočetní čas. Pro obecný druh zadání naše nová heuristika nedosahovala nejlepších výsledků ve výpočetním čase, ale pro konkrétní typ zadání, který jsme předpokládali, se jevila jasně lepší v obou sledovaných veličinách. Jedná se o zadání, kde je vynucena velká interakce mezi jednotlivými agenty a zároveň jsou všechny cílové pozice umístěné poblíž sebe.

Z vlastností nově navržené heuristiky, jsme také odvodili teoretické vylepšení algoritmu CBM, který v roce 2016 představili autoři Hang Ma a Sven Koenig[10]. Toto vylepšení spočívalo ve zmenšení grafu, na kterém se až exponenciálně krát hledá tok, za cenu jednoho průchodu do hloubky daného grafu na konci výpočtu.

7.1 Budoucí práce

Jelikož výpočet naší nově nevržené heuristiky neni úplně přímočarý a triviální, je zde větší prostor pro vylepšování implementace. Některé tyto vylepšení jsme v práci samotné popsali a začlenili do naší iplementace. Například se jedná o chytré stavění časově expandovaného grafu a využívání již jednou postavených částí. Dalším důležitým prvkem heuristiky je hledání toku, o kterém jsme se ale blíže nezmiňovali. Zajímavým experimentem by bylo porovnání jednotlivých algoritmů na hledání maximálního toku v síti. Navíc je možnost provést teoretický odhad časové složitosti těchto algoritmů, který by mohl vycházet značně lépe, než odhad pro obecné sítě, protože náš časově expandovaný graf má specifickou strukturu.

Známé algoritmy pro hledání maximálního toku také mohou začít s libovolným tokem[11]. Dalším možným vylepšením je tedy nezačínat s tokem nulovým, ale využít maximální nalezený tok v předchozí iteraci heuristiky.

Jednou z velkých oblastí a potenciálního vylepšení je využití paralelizace. V dnešní době jsou vícejádrové procesory prakticky samozřejmostí a protože při experiemntech byl vždy úzkým hrdlem výpočetní čas a nikoliv paměť, jeví se paralelizace jako dobrý kandidát na vylepšení. Obtížným úkolem ovšem bude realizace paralelního prohledávání stavového prostoru, určení správných míst pro větvení a navržení podmínek pro ukončení prohledávání oblastí, které mají špatné vyhlídky na to být úspěšné.

V neposlední řadě je zajímavou úlohou experimentálně ověřit naše navrhované vylepšení algoritmu CBM a naměřit změnu výpočetního času při zmenšení grafu.

Literatura

[1] Adi Botea and Pavel Surynek. Multi-agent path finding on strongly biconnected digraphs. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 2024–2030, 2015.

[2] Edsger. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[3] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, 1972.

[4] Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976.

[5] L. R. Ford and D. R. Fulkerson. Maximal Flow through a Network. *Canadian Journal of Mathematics*, 8:399–404.

[6] Kikuo Fujimura. *Motion planning in dynamic environments.* Springer-Verlag, 1991.

[7] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics*, 4(2):100–107, 1968.

[8] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[9] Ryan Luna and Kostas E. Bekris. Push and swap: Fast cooperative path-finding with completeness guarantees. In Toby Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 294–300. IJCAI/AAAI, 2011.

[10] Hang Ma and Sven Koenig. Optimal target assignment and path finding for teams of agents. In Catholijn M. Jonker, Stacy Marsella, John Thangarajah, and Karl Tuyls, editors, *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016*, pages 1144–1152. ACM, 2016.

[11] Martin Mareš. *Krajinou grafových algoritmů.* ITI series. Institut Teoretické Informatiky, 2015.

[12] Judea Pearl. *Heuristics - intelligent search strategies for computer problem solving.* Addison-Wesley series in artificial intelligence. Addison-Wesley, 1984.

[13] D. Pottinger. Implementing coordinated movement. *Game Developer Magazine*, 1999.

[14] Daniel Ratner and Manfred K. Warmuth. Finding a shortest solution for the N x N extension of the 15-puzzle is intractable. In Tom Kehler, editor, *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, August 11-15, 1986. Volume 1: Science.*, pages 168–172. Morgan Kaufmann, 1986.

[15] Stuart J. Russell and Peter Norvig. *Artificial intelligence - a modern approach: the intelligent agent book*. Prentice Hall series in artificial intelligence. Prentice Hall, 1995.

[16] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent path finding. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.* AAAI Press, 2012.

[17] David Silver. Cooperative pathfinding. In R. Michael Young and John E. Laird, editors, *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, June 1-5, 2005, Marina del Rey, California, USA*, pages 117–122. AAAI Press, 2005.

[18] B. Stout. Smart moves: Intelligent pathfinding. *Game Developer Magazine*, 1996.

[19] Pavel Surynek. A novel approach to path planning for multiple robots in bi-connected graphs. In *2009 IEEE International Conference on Robotics and Automation, ICRA 2009, Kobe, Japan, May 12-17, 2009*, pages 3613–3619. IEEE, 2009.

[20] Pavel Surynek. An optimization variant of multi-robot path planning is intractable. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010.

[21] Jur van den Berg, Jack Snoeyink, Ming C. Lin, and Dinesh Manocha. Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In Jeff Trinkle, Yoky Matsuoka, and José A. Castellanos, editors, *Robotics: Science and Systems V, University of Washington, Seattle, USA, June 28 - July 1, 2009*. The MIT Press, 2009.

[22] Jur P. van den Berg and Mark H. Overmars. Prioritized motion planning for multiple robots. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, Edmonton, Alberta, Canada, August 2-6, 2005*, pages 430–435. IEEE, 2005.

[23] Ko-Hsin Cindy Wang and Adi Botea. Tractable multi-agent path planning on grid maps. In Craig Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 1870–1875, 2009.

[24] Gerhard Weiss. *Multiagent Systems*. Intelligent Robotics and Autonomous Agents. MIT Press, Cambridge, MA, 2013.

[25] Michael J. Wooldridge. *An Introduction to MultiAgent Systems (2. ed.).* Wiley, 2009.

[26] Jingjin Yu and Steven M. LaValle. Multi-agent path planning and network flow. *CoRR*, abs/1204.5717, 2012.

[27] Jingjin Yu and Steven M. LaValle. Planning optimal paths for multiple robots on graphs. In *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*, pages 3612–3617. IEEE, 2013.

[28] Jingjin Yu and Steven M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In Marie desJardins and Michael L. Littman, editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*. AAAI Press, 2013.

Seznam obrázků

Přílohy

Implementační detaily

V této příloze si popíšeme některé datové struktury a procedury použité při implementaci algoritmu A*. Jak bylo dříve zmíněno, řešení hledáme v prostoru stavů. Při tomto prohledávání si stavíme strom stavů, kde vrchol odpovídající stavu α_i má potomky odpovídající možným stavům α_{i+1}. Pro takovýto vrchol si pamatujeme kromě příslušného stavu i hodnoty funkcí g a $f = g + h$ a předchůdce vrcholu. Potomky vrcholů si nemusíme pamatovat, jelikož nikdy během výpočtu neprocházíme strom směrem od kořenu, pouze přidáváme nově nalezené stavy nebo přesouváme již existující pod vrcholy, ze kterých existuje lepší cesta vzhledem k g.

Samotné vrcholy neudržujeme ve stromové struktuře, nýbrž je rozdělíme na dvě disjunktní množiny - otevřené a uzavřené. S uzavřenými vrcholy již nemusíme pracovat díky konzistenci použitých heuristik. Při přechodu do nového stavu musíme kontrolovat, zda tento stav nebyl již dříve uzavřený. Uzavřené vrcholy si tedy budeme udržovat v hash tabulce, oproti které všechny nové stavy porovnáváme. Pokud je takovýto stav již uzavřený, nemusíme jej dále uvažovat. Do této tabulky pouze přidáváme nové vrcholy, nikdy je nemažeme.

Obdobným způsobem porovnáváme stavy i s otevřenými vrcholy, tedy si i pro ně pořídíme hash tabulku. Z té ovšem vybíráme vrchol s nejnižší hodnotou f. Pro tuto operaci si pořídíme haldu s ukazateli na všechny otevřené vrcholy seřazenou podle hodnoty f. Po vybrání minima z haldy vymažeme příslušný vrchol i z hash tabulky. Otevřené vrcholy mohou navíc měnit hodnotu funkce f, pokud nalezneme lepší cestu do odpovídajícího stavu a tím snížíme hodnotu g. V takovém případně upravíme hodnoty g a f daného vrcholu, změníme jeho předchůdce a provedeme příslušné operace na haldě k udržení podmínky haldy.

Použitá hash funkce je následující

$$hash(\overrightarrow{x}) = \sum_{i=1}^{n} i * x_i, \qquad \text{kde } n \text{ je počet vrcholů.}$$

$$x_i = \begin{cases} 0, & \text{pro neobsazený vrchol,} \\ \text{číslo agenta,} & \text{jinak.} \end{cases}$$

Takováto hash funkce zajisté nezabraňuje kolizím a musíme tedy jednotlivě kontrolovat všechny vrcholy odpovídající $hash(\overrightarrow{x})$. Při testování se ovšem ukázalo, že čas strávený procházením kolizních stavů je zanedbatelný v rámci celkového výpočetního času. Jedna z možných lepších hashovacích funkcí je

$$hash(\overrightarrow{x}) = \prod_{i=1}^{n} p_i^{x_i}, \qquad \text{kde } p_i \text{ je } i\text{-té prvočíslo a } n \text{ počet vrcholů.}$$

$$x_i = \begin{cases} 0, & \text{pro neobsazený vrchol,} \\ \text{číslo agenta,} & \text{jinak.} \end{cases}$$

Takováto funkce dává jednoznačné zakódování, jelikož každé má unikátní rozložení na prvočísla. Výpočet je ovšem značně složitější použitím mocnění a potřebou znát alespoň tolik prvočísel, kolik je vrcholů v původním grafu.

Pro použité heuristiky je potřeba znát vzdálenost všech dvojic vrcholů. Tu jednou spočteme před začátkem výpočtu pomocí Floydova–Warshallova algoritmu a v průběhu pouze vyčítáme potřebné hodnoty ze zapamatované tabulky. Dále je pro naši heuristiku potřeba umět nalézt největší tok v síti. K tomuto účelu jsme zvolili Edmondsův-Karpův algoritmus[3], který běží v čase $\mathcal{O}(EF)$, kde E je počet hran a F je maximální tok. Navíc F bude vždy maximálně počet agentů. To je lepší odhad, než například u Dinicova algoritmu, který nad grafy s jednotkovou kapacitou běží v čase $\mathcal{O}(V^{2/3}E)$[11].

Vstupní formát

Náš program, na přiloženém médiu přeložený a pojmenovaný `AStar`, bere pět různých parametrů. První bezargumentový parametr je *-h*. Pokud je tento přítomný, jsou všechny ostatní argumenty ignorovány, dojde k vypsání nápovědy a ukončení programu. Další parametr je *-f*, ten navíc bere jako jeden argument jméno souboru obsahujícího zadání úlohy. Pokud option *-f* není přítomný, pak se zadání úlohy načítá ze standardního vstupu. Jediný povinný parametr je *-e*, který bere jeden argument. Tento argument určuje, která heuristika se má při výpočtu použít. Možné volby jsou:

- `short` pro heuristiku nejdelší z nejkratších cest.
- `flow` pro heuristiku využívající tok v sítích.

Dalším parametrem je *-a*, který, pokud je přítomný, přepíná úlohu na anonymní multi-agentní problém. V tomto problému nemá každý agent přiřazený vlastní cíl, kam má dojít, pouze se požaduje, aby na konci byl každý cíl obsazený nějakým agentem. Pokud parametr není přítomný, řeší se klasická úloha. Oba druhy úloh přijímají stejně formátované zadání.

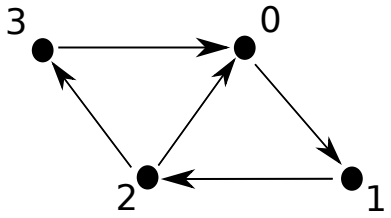
Poslední bezargumentový option je *-g*. Pokud je tento přítomný, předpokládáme, že vstupním grafem bude mřížka n krát n. V tomto grafu je každý vrchol spojený neorientovanou hranou se všemi svými sousedy, pokud tímto sousedem není stěna. Do vrcholu označeného jako stěna nevede žádná hrana. Tento typ grafu vede k jednoduššímu zapsání. Vrcholy jsou označeny `.`, stěny `x` a počáteční polohu agenta reprezentuje jeho číslo na místě vrcholu (agenti jsou číslováni od 0). Vstup úlohy se pak skládá z řádku, který obsahuje dvě čísla - počet agentů a délka hrany mřížky. Další řádek obsahuje cíle agentů, kde j-té číslo je cíl j-tého agenta. Vrcholy se číslují od nuly po řádcích. Následuje výše popsaný graf se stěnami a počátečními pozicemi agentů (viz Obrázek 7.1).

```
2 4
11 15
. . 0 1
. x x x
. x . .
. . . .
```

Obrázek 7.1: **Ukázka vstupu mřížky.**

Pokud není option *-g* přítomný, pak se předpokládá, že je úloha nad obecným grafem. Tato úloha má následující vstupní formát. První řádek obsahuje dvě čísla - počet agentů a počet vrcholů. Agenti i vrcholy jsou číslováni od 0. Další řádek obsahuje cíle agentů. Opět pro každého agenta jedno číslo, které odpovídá číslu vrcholu, který je jeho cíl. Na dalším řádku je počáteční stav - pro každý vrchol jedno číslo. Pokud je v počátečním stavu vrchol neobsazený, pak je toto číslo -1, jinak je to číslo agenta, který se v tomto vrcholu nachází. Následuje tolik řádků, kolik je vrcholů. Na každém řádku je seznam sousedů tohoto vrcholu. To odpovídá orientovaným hranám z vrcholu do všech jeho sousedů. Pro jednoduchý

graf z Obrázku 7.2, kde agent 0 začíná ve vrcholu 2 a končí ve vrcholu 1, agent 1 začíná ve vrcholu 0 a končí ve vrcholu 2 a agent 2 začíná ve vrcholu 1 a končí ve vrcholu 0, odpovídá vstup na Obrázku 7.3.



Obrázek 7.2: **Obecný graf.**

```
3 4
1 2 0
1 2 0 -1
1
2
0 3
0
```

Obrázek 7.3: **Ukázka vstupu obecného grafu.**

Výstupní formát

Během načítání parametrů a vstupu úlohy program hlásí pokrok, případně objevenou chybu, která vždy způsobí ukončení programu. V případě bezchybného zadání úlohy dojde k vlastnímu výpočtu, jehož začátek je ohlášen. Během výpočtu k dalším výpisům nedochází, až po skončení je ohlášeno, zda existuje řešení, v tom případě je i vypsáno, nebo zda řešení neexistuje. Řešení je vypsáno po řádcích, kde každý řádek odpovídá jinému stavu grafu. Jednotlivé řádky se liší krokem právě jednoho agenta, který se přesunul. Na začátku řádku je vypsáno, ke kterému časovému intervalu tento stav patří. Dále dostáváme informaci o délce makespanu potřebnému k vyřešení úlohy a kolik stavů se při hledání řešení navštívilo. Příklad výstupu pro vstup z Obrázku 7.3 je vidět na Obrázku 7.4.

```
> ./AStar -e short -f example_input
short - Using shortest path heuristic
Loading input...
Done loading input
Solving...
Number of visited states: 11
Solution:
8: 2 0 1 -1
7: -1 0 1 2
6: 0 -1 1 2
5: 0 1 -1 2
4: 0 1 2 -1
3: -1 1 2 0
2: 1 -1 2 0
1: 1 2 -1 0
1: 1 2 0 -1
makespan: 8
```

Obrázek 7.4: **Ukázka výstupu programu.**

Generátory

Pro účely testování jsme vytvořili několik generátorů, které jsou použity pro generování různých typů grafů. Všechny níže popsané generátory jsou také na přiloženém médiu.

Prvním je generátor, který vytváří čistě orientované silně souvislé grafy. Toho lze docílit vytvořením orientované kružnice a následným přidáváním orientovaných uší. Počáteční rozmístění agentů i jejich cílový stav je náhodně určen. Generátor samotný bere čtyři argumenty - počet agentů, velikost počátečního cyklu, velikost uší a počet uší. Výhodou tohoto typu grafů je, že pokud je počet agentů alespoň o dva menší než celkový počet vrcholů, pak je úloha nad tímto grafem vždy řešitelná [1].

Dalším je generátor mřížkových grafů. Ten generuje grafy s dvěma typy překážek - úzké hrdlo a překážka uprostřed mřížky (viz Obrázek 6.2). Podle parametrů generátoru jsou počáteční pozice agentů buďto náhodné nebo v jedné oblasti grafu. Obdobně cílové pozice mohou být náhodné nebo v jedné jiné oblasti grafu. Velikost hrany mřížky může být zvolena z množiny 5, 7, 10 a 13. Počet agentů je také volitelný, předpokládá se ovšem, že takový počet agentů se vejde do zvolené mřížky. Vygenerovaný graf je ve formátu pro mřížkové grafy popsaný dříve.