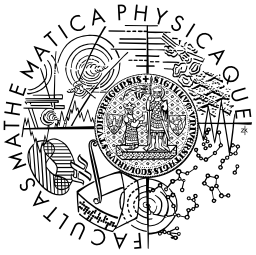


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

**DIPLOMOVÁ PRÁCE**

Michal Nekvinda

# Hledání robustních cest pro více agentů

Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: prof. RNDr. Roman Barták, Ph.D.
Studijní program: Informatika
Studijní obor: Umělá inteligence

Praha 2020

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

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

V ..... dne .....  .....

Podpis autora

Rád bych poděkoval prof. RNDr. Romanu Bartákovi, Ph.D. za vedení této práce a za cenné rady a komentáře, které mi během psaní práce poskytl. Dále děkuji své rodině za trpělivost a podporu během mého studia.

Název práce: Hledání robustních cest pro více agentů

Autor: Michal Nekvinda

Katedra: Katedra teoretické informatiky a matematické logiky

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

Abstrakt: Práce se věnuje hledání robustních nekonfliktních cest v multi-agent path finding (MAPF). Představíme zde několik nových technik pro konstrukci těchto cest a popíšeme jejich vlastnosti. Budeme se zabývat použitím techniky plánování s alternativami, na základě níž vytvoříme pro agenty stromový plán, přičemž konkrétní průchod si agenti zvolí na základě aktuálního zpoždění během cesty. Dále představíme algoritmus zvyšující robustnost při zachování původní délky řešení a zkombinujeme ho s předchozím přístupem. Věnovat se budeme také metodě zvyšující robustnost pomocí změn rychlosti agentů. Následně experimentálně ověříme použitelnost všech technik na různých typech grafů. Ukážeme, že navržené metody jsou výrazně robustnější než klasické řešení a jisté výhody mají i oproti doposud známým konstrukcím robustních plánů.

Klíčová slova: MAPF, robustnost, plánování s alternativami, změna rychlosti

Title: Robust multi-agent path finding

Author: Michal Nekvinda

Department: Department of Theoretical Computer Science and Mathematical Logic

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

Abstract: The thesis is devoted to finding robust non-conflict paths in multi-agent path finding (MAPF). We propose several new techniques for the construction of these types of paths and describe their properties. We deal with the use of contingency planning and we create a tree plan for the agents where the specific path is chosen by the agents during the execution based on the current delay. Next we present an algorithm that increases robustness while maintaining the original length of the solution and we combine it with the previous approach. Then we will focus on the method of increasing robustness by changing the speed of agents. Finally we experimentally verify the applicability of these techniques on different types of graphs. We will show that all the proposed methods are significantly more robust than the classic solution and they also have certain advantages over previously known constructions of robust plans.

Keywords: MAPF, robustness, contingency planning, speed change

# Obsah

# Úvod

V posledních letech se na poli umělé inteligence můžeme setkat s rostoucím zájmem o oblast multi-agent path finding (MAPF), neboli hledání cest v prostředí s více agenty. Úkolem je najít pro každého agenta v daném prostředí cestu z jeho počáteční pozice do cíle tak, aby se žádní dva agenti po cestě nesrazili. Navíc bychom rádi našli takové cesty, které jsou z hlediska námi vybraného kritéria co nejlepší.

Tato oblast má široké praktické využití zasahující do oblastí každodenní činnosti. Setkat se s ní můžeme při řízení přepravních robotů ve skladech (Wurman a kol., 2008), při navigaci průjezdů vozidel přes autonomní křižovatky (Dresner a Stone, 2008), až po pojíždění letadel na letištích (Morris a kol., 2016). Ve virtuálním světě najdeme uplatnění přístupů MAPF například v oblasti počítačových her (Ma a kol., 2017).

Postupně byly vyvinuty různé algoritmy a teoretické přístupy pro nalezení optimálních cest jednotlivých agentů. V praxi však dochází vlivem vnějších jevů prostředí (ucpaná křižovatka, chodci na přechodu) a také nedokonalostí agentů samotných (desynchronizace, konstrukční chyby) k odchylkám od zamýšlené podoby plánu. Ty se projevují zpožděním oproti ideálnímu plánu a často znamenají kolizi agentů, která způsobí nedokončení jejich cesty. Na řadu pak přichází použití robustních plánů. Pomocí nich se snažíme navrhnout cesty pro agenty tak, aby byly připravené na možnost výskytu drobných anomálií a dokázaly se s nastalým zpožděním agentů vypořádat. Za cenu mírného zvýšení délky optimální cesty snižujeme pravděpodobnost kolize během provádění plánu a zvyšujeme šanci jeho úspěšného provedení. Postupem času se vyvíjejí určité techniky, které se o konstrukci robustních plánů pokouší.

Cílem práce je představit nové způsoby plánování robustních cest, které oproti řešení klasického MAPF problému vykazují větší míru odolnosti vůči kolizím a jsou schopné vypořádat se s opakovanými výskyty zpoždění jednotlivých agentů během provádění plánu.

Na začátku práce se blíže seznámíme s terminologií z oblasti MAPF a zavedeme některé základní pojmy typické pro tuto oblast. Dále se seznámíme s pojmy týkající se robustnosti a ukážeme dosud známé přístupy pro tvorbu robustních cest. Poté přejdeme k bližšímu popisu technik a algoritmů, které nám umožňují hledat nekonfliktní optimální cesty v prostředí více agentů. Zde se také dozvíme, že tento problém je v obecné podobě těžký.

V následujících několika kapitolách představíme nové způsoby hledání robustních cest. Zavedeme vlastní definice robustnosti a implementujeme je do klasického MAPF problému. Z počátku se budeme věnovat především technice plánování s alternativami, v anglické literatuře známé pod názvem *contingency planning*. S ní se setkáváme zejména v oblasti plánování a rozvrhování v nedeterministickém prostředí. Hlavní myšlenkou je nahrazení prostých sekvenčních cest plánem ve formě stromu, který nazveme plánem s alternativami. Ten obsahuje více možností provedení, přičemž konkrétní podoba závisí na aktuálních podmínkách v prostředí. Protože nalezení cesty agenta v MAPF můžeme chápat jako úlohu ve smyslu klasického plánování s akcemi typu *přejeď z místa A do*

*místa B* a výskyt zpoždění jako neočekávanou akci *stůj*, vypadá použití nástroje z oblasti plánování relativně přímočaře. Při jeho aplikaci do MAPF se budeme blíže zabývat výběrem míst, ze kterých alternativní plány vést a možnostmi řešení konfliktů mezi alternativami a původním cestami. Výsledkem bude několik definic robustních plánů a návodů na jejich konstrukci.

Vzhledem k tomu, že míra úspěšnosti předcházení kolizím do jisté míry závisí na kvalitě počátečního sekvenčního plánu, ze kterého alternativy vedeme, podíváme se rovněž i na možnosti jeho vylepšení. Na základě této motivace popíšeme polynomiální algoritmus, který umožní navýšit robustnost bez prodloužení maximální délky plánu ve vybraném MAPF problému a použijeme ho jako pomocnou úpravu k plánům s alternativami. Tento způsob, nazvaný jako zvýšení robustnosti pomocí přidání *wait* akcí, bude však mít univerzálnější využití a uspokojivé výsledky z hlediska robustnosti může dávat i sám o sobě, při aplikaci na libovolnou množinu validních plánů.

Dále se seznámíme s přístupem, který zvyšuje robustnost pomocí změn rychlosti agentů. Na rozdíl od všech dříve známých konstrukcí robustních plánů zde budeme schopni aktivně eliminovat vzniklé zpoždění, takže i přes jeho opakované výskyty dokážeme agenty dostat do cíle v čase, který jsme původně naplánovali.

Všechny uvedené metody podrobíme testování a srovnáme jejich vlastnosti a úspěšnost. Přidáme srovnání vůči klasickému validnímu řešení i dalším, dosud známým technikám pro konstrukci robustních plánů. Experimenty provedeme pomocí počítačové simulace na několika typech grafů s použitím známých benchmarků vytvořených právě pro oblast MAPF. K testování použijeme námi vytvořený simulátor, který kromě měření sledovaných hodnot umožňuje rovněž grafickou vizualizaci různých typů robustních plánů.

# 1. Základní pojmy z oblasti MAPF

Na začátek si představíme několik klíčových definic a pojmů z oblasti MAPF a zavedeme značení, jež budeme nadále v práci používat. Vycházíme z terminologie popsané v článku Stern a kol. (2019).

## 1.1 MAPF problém

Jako klasický MAPF problém s $k$ agenty označujeme trojici $\langle G, s, t\rangle$, kde:

- $\mathbf{G} = (V, E)$ je neorientovaný graf,
- $\mathbf{s} : [1,...,k] \longrightarrow V$ je funkce, která každému agentovi přiřazuje jeho počáteční vrchol,
- $\mathbf{t} : [1,...,k] \longrightarrow V$ je funkce, která každému agentovi přiřazuje jeho cílový vrchol.

Dále předpokládáme, že čas je diskrétní a můžeme ho rozdělit do jednotlivých úseků. V každém z nich se každý agent nachází v právě jednom vrcholu grafu $G$.

Agenti provedou za jednotku času právě jednu akci, přičemž na výběr máme celkem dva typy akcí. První možností je přesun do libovolného sousedního vrcholu (akce *move*), druhým typem je akce *wait*, neboli čekání v daném vrcholu. Formálně můžeme akci agenta vyjádřit funkcí $a : V \longrightarrow V$, tedy zápis $a(v) = v'$ znamená, že po provedení akce $a$ se agent přesune z vrcholu $v$ do vrcholu $v'$.

**Definice 1** (poloha agenta po $k$-krocích)**.** *Pro posloupnost akcí $A = (a_1, a_2,..., a_n)$ agenta $i$, označme $\pi_i[k]$ vrchol, do kterého se agent $i$ dostane z vrcholu $s(i)$ po provedení prvních $k$ akcí z posloupnosti $A$.*

**Definice 2** (plán pro jednoho agenta)**.** *Posloupnost akcí $A$ nazveme plánem pro agenta $i$, pokud se vykonáním těchto akcí dostaneme z vrcholu $s(i)$ do vrcholu $t(i)$. Platí tedy, že $\pi_i[|A|] = t(i)$. Plán agenta $i$ označujeme symbolem $\pi_i$. Jeho délkou pak rozumíme počet akcí, které agent vykonal při přemístění z vrcholu s(i) do vrcholu t(i).*

**Definice 3** (řešení MAPF problému)**.** *Řešením problému MAPF s $k$ agenty je množina $k$ plánů jednotlivých agentů.*

Pokud máme zadefinovaný MAPF problém s $k$ agenty a mluvíme o plánu $\pi$, máme tím na mysli řešení tohoto MAPF problému s plány jednotlivých agentů. Tedy označujeme $\pi = \{\pi_1,..., \pi_k\}$.

## 1.2 Konflikty mezi agenty

Pro nalezené řešení MAPF chceme, aby bylo bezpečně proveditelné a všichni agenti se dostali bez srážky do cíle. Existují různé typy konfliktů, které mohou

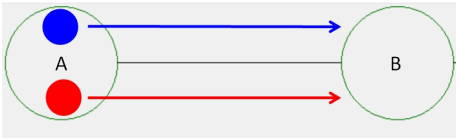
úspěšné vykonání plánů ohrozit. Při hledání řešení bychom si měli stanovit, které konflikty jsou pro nás fatální a v plánech je zakázat. S rostoucími nároky na omezení různých konfliktů se samozřejmě snižuje množství přípustných řešení. Řešení MAPF, které je bez konfliktů, nazýváme též validní.

Podívejme se nyní, jak přísně se dá při zákazu různých forem konfliktů postupovat. Nejmírnější forma restrikce v sobě zahrnuje omezení pouze hranových konfliktů.

### 1.2.1 Hranový konflikt

**Definice 4** (hranový konflikt)**.** *Plány agentů $\pi_i$ a $\pi_j$ obsahují* ***hranový konflikt****, pokud existuje $k$ takové, že $\pi_i[k] = \pi_j[k]$ a zároveň $\pi_i[k+1] = \pi_j[k+1]$. Tedy nastane situace, kdy dva agenti jedou ze stejného vrcholu v tentýž čas po stejné hraně týmž směrem.*

Ilustraci hranového konfliktu dvou agentů vidíme na obrázku 1.1.




Obrázek 1.1: Hranový konflikt dvou agentů na hraně mezi vrcholy $A$ a $B$.

Jak je nám známo, tak ve všech pracích, zabývajících se doposud tímto tématem, se setkáme i s omezením vrcholových konfliktů.

### 1.2.2 Vrcholový konflikt

**Definice 5** (vrcholový konflikt)**.** *Plány agentů $\pi_i$ a $\pi_j$ obsahují* ***vrcholový konflikt****, pokud existuje $k$ takové, že $\pi_i[k] = \pi_j[k]$. Tedy nastane situace, kdy se dva agenti potkají ve stejném vrcholu.*

Pro vrcholový a hranový konflikt platí následující jednoduché tvrzení. Důkaz plyne přímo z příslušných definic.

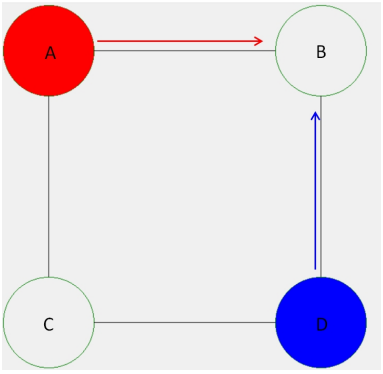
**Tvrzení 1.** *Jestliže plán nemá vrcholový konflikt, pak nemá ani hranový konflikt.*

Ukázku vrcholového konfliktu dvou agentů vidíme na obrázku 1.2.

V souvislosti s vrcholovým konfliktem musíme ještě upřesnit chování agentů po dosažení cílového vrcholu. V literatuře se setkáme se dvěma přístupy. První z nich předpokládá setrvání agenta v cíli, kde může být i nadále účastníkem vrcholových konfliktů. V angličtině se používá výraz *stay at target*.

Druhý přístup počítá s odstraněním agenta z grafu po provedení všech akcí jeho plánu. V anglické terminologii označujeme toto chování pojmem *disappear at target*. Odpovídá konceptu soukromého parkovacího místa, které má agent vyhrazené ve svém cíli. Na tomto místě už další vrcholové konflikty způsobit nemůže.

Ve většině publikací z oboru MAPF se setkáváme spíše s přístupem *stay at target*, existují ale i práce zabývající druhým případem (Ma a kol., 2019). My




Obrázek 1.2: Vrcholový konflikt dvou agentů ve vrcholu $B$.

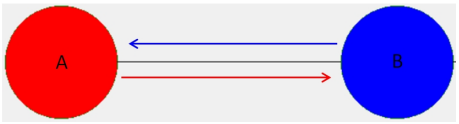
v této práci, nebude-li uvedeno jinak, předpokládáme variantu první, tedy *stay at target*.

Kromě dvou výše zmíněných základních konfliktů byly popsány a definovány ještě některé další, které může mít smysl v určitých případech zakázat. V této práci dále pracujeme i se zákazem tzv. vzájemné výměny vrcholů (*swapping conflict*).

### 1.2.3 Konflikt vzájemné výměny vrcholů

**Definice 6** (konflikt vzájemné výměny vrcholů)**.** *V plánech dvou agentů $\pi_i$ a $\pi_j$ dochází ke* ***konfliktu vzájemné výměny vrcholů****, pokud existuje $k$ takové, že $\pi_i[k+1] = \pi_j[k]$ a zároveň $\pi_i[k] = \pi_j[k+1]$. Tedy nastane situace, kdy si dva agenti během jednoho kroku vymění své pozice.*

Ilustrace konfliktu vzájemné výměny vrcholů vidíme na obrázku 1.3.




Obrázek 1.3: Konflikt vzájemné výměny vrcholů dvou agentů mezi vrcholy $A$ a $B$.

Jinými slovy řečeno, dva agenti během stejného časového úseku přejíždějí po stejné hraně. Někde na této hraně pak reálně nastává srážka. Zákaz konfliktu vzájemné výměny vrcholů bývá relativně běžný, nicméně může dávat dobrý smysl ho v plánech dovolit (Ma a kol., 2016). V reálné situaci se to týká případů, kdy máme dva uzly spojené cestou, která umožňuje obousměrný provoz.

V dalším průběhu budeme používat následující definici pro validní řešení.

**Definice 7** (validní řešení)**.** *Řešení MAPF problému s $k$ agenty nazveme* ***validní****, pokud se v žádných dvou plánech $\pi_i$, $\pi_j$ nevyskytuje ani vrcholový konflikt, ani konflikt vzájemné výměny vrcholů.*

Obdobně budeme pro vybraný MAPF problém s $k$ agenty užívat pojem validní plán $\pi$, čímž máme opět na mysli validní řešení obsahující množinu $k$ plánů jednotlivých agentů. Naopak výskyt konfliktu v plánu bude odteď značit buď vrcholový konflikt, nebo konflikt vzájemné výměny vrcholů.

## 1.3 Účelová funkce

Při hledání řešení MAPF problému často nechceme jen plán $\pi = \{\pi_1,..., \pi_k\}$, který je validní, ale požadujeme, aby byl v určitém ohledu co nejlepší možný. Snažíme se najít takové řešení, které by minimalizovalo (maximalizovalo) hodnotu námi zvolené účelové funkce. V literatuře se nejvíce setkáme s použitím funkcí *makespan* a *sum of costs* ($SoC$). Pojďme se proto blíže podívat na to, co konkrétně vyjadřují. Označením $|\pi_i|$ budeme rozumět délku plánu $\pi_i$.

- **Makespan** – množství času, které agenti potřebují k přesunu ze svých počátečních pozic do cílových. Formálně vyjádřeno $makespan_\pi = max_i|\pi_i|$.

  Makespan se hodí v problémech, u kterých potřebujeme omezit celkovou dobu trvání a platíme za spotřebovaný čas.

- **SoC** – součet časů, které jednotliví agenti potřebují k přesunu ze svých počátečních pozic do cílových. Tedy $SoC_\pi = \sum_i |\pi_i|$.

  SoC využijeme u problémů, kde chceme omezit délky cest. Důležitá pro nás je cena, kterou platíme za provedené akce, či ujeté vzdálenosti.

V rámci této práce budeme kvalitu plánů primárně posuzovat podle hodnoty účelové funkce makespan.

## 1.4 Pohyb agenta v grafu

Vzhledem k oblastem, ve kterých se MAPF používá, bývá zvykem zaměřit se při řešení problémů na grafy, které jsou rovinné. Tento předpoklad není nezbytně nutný a všechny dále představené techniky hledání (robustních) řešení lze aplikovat i na grafy, které rovinné nejsou. Požadujeme pouze to, aby se žádné dvě hrany grafu nekřížily jinde než ve vrcholech, protože na těchto průsečících by mohlo docházet ke kolizím.

Dále z definice polohy agenta po $k$ krocích (definice 1) a z důvodu zavedení diskrétního času předpokládáme přítomnost agenta v nějakém vrcholu v každém diskrétním časovém úseku. Přejezd po hraně mezi sousedními vrcholy se proto musí odehrát vždy mezi dvěma časovými úseky a pokaždé se stejnou délkou trvání. Z tohoto důvodu je rozumné předpokládat, že všechny hrany grafu jsou přibližně stejně dlouhé.

# 2. Robustnost MAPF

Nechť máme definovaný libovolný MAPF problém, pro který najdeme validní řešení. Potom přirozeně chceme toto řešení provést, tj. aplikovat jej na agentech. V klasickém MAPF předpokládáme, že provedení plánu proběhne přesně podle předpokladů. Bohužel se vlivem různých nedokonalostí jednotlivých agentů či prostředí mohou začít vyskytovat odchylky od teoretické podoby plánu, vyjádřené formou zpoždění agentů. Následkem toho pak může dojít ke kolizi.

**Definice 8** (kolize agentů)**.** *Řekneme, že při vykonávání validního plánu došlo ke kolizi dvou agentů, pokud nastal vlivem jejich zpoždění vrcholový konflikt nebo konflikt vzájemné výměny vrcholů.*

Nyní nám ještě zbývá formálně popsat pojem zpoždění. Při jeho zavedení využijeme definici, kterou popsali Atzmon a kol. (2018). Začneme nejprve s plánem jednoho agenta. Zpoždění v plánu $\pi_i$ definujeme jako trojici $D = \langle \pi_i, i, t_D \rangle$ a chápeme ho tak, že agent $i$ nevykonal akci přesunu, kterou měl ve svém plánu $\pi_i$ a místo toho zůstal v čase $t_D$ stát na pozici, ve které byl v čase $t_D - 1$. Zpožděný agent se začíná nacházet v místech, kde už původně být neměl, což se stává zdrojem kolizí s ostatními agenty.

Přidáním zpoždění $D = \langle \pi_i, i, t_D \rangle$ do plánu $\pi_i$, dostaneme plán $D[\pi_i]$, který se oproti původnímu plánu liší takto:

$$D[\pi_i][t] = \begin{cases} s(i), & \text{pro t} = 0, \\ \pi_i[t] & \text{pro } t < t_D, \\ \pi_i[t-1] & \text{jinak.} \end{cases}$$

Aplikujeme-li na plán $\pi_i$ množinu zpoždění $\mathcal{D}_i = \{D_1,..., D_n\}$, obdržíme plán $\mathcal{D}_i[\pi_i] = D_n[D_{n-1}[...[D_1[\pi_i]]...]$. Rozšířením pojmu na $k$ agentů v MAPF problému nakonec dostaneme plán $\mathcal{D}[\pi] = \{\mathcal{D}_1[\pi_1], ..., \mathcal{D}_k[\pi_k]\}$.

Plán $\pi = \{\pi_1,..., \pi_k\}$ je *robustní* vůči zpoždění $\mathcal{D} = \{\mathcal{D}_1,..., \mathcal{D}_k\}$, pokud je plán $\mathcal{D}[\pi] = \{\mathcal{D}_1[\pi_1],..., \mathcal{D}_k[\pi_k]\}$ validní vůči libovolné podmnožině zpoždění $\mathcal{D}_i$ pro každé $i = 1,..., k$. Jinými slovy můžeme říci, že robustní plán je nějakým způsobem odolný vůči výskytu nenadálých situací, po kterých mohou agenti pokračovat ve vykonávání svého plánu, aniž by se objevila kolize.

Daný pojem robustnosti nijak striktně nespecifikuje požadavky, které by měl plán po formální stránce splňovat. K robustnosti v plánu můžeme přistoupit různou formou. V rámci každé z nich zavádíme určitá pravidla a omezení, která nám umožní tento konkrétní druh robustních plánů vytvářet. Z něj následně blíže vyplývá, pro jaké množiny zpoždění $\mathcal{D}$ zůstává plán validní. Nyní se podíváme na dvě konkrétní formy robustnosti, které byly doposud představeny.

## 2.1 *K*-robustnost

První možnost konstrukce robustních plánů nám dává pojem $k$-robustnosti, představený v článku Atzmon a kol. (2018). Před definicí samotného pojmu nejdříve uveďme jednu pomocnou definici.

**Definice 9** ($k$-zpožděný konflikt)**.** *Plán $\pi$ má $k$-zpožděný konflikt $\langle i, j, t\rangle$ pokud existuje $\Delta \in \langle 0, k\rangle$ takové, že $\pi_i[t] = \pi_j[t + \Delta]$.*

Nyní nám již nic nebrání v definici samotného pojmu.

**Definice 10** ($k$-robustnost)**.** *Plán $\pi$ nazveme **$k$-robustní**, pokud neobsahuje žádný $k$-zpožděný konflikt.*

Najdeme-li plán, který je $k$-robustní dle definice výše, znamená to, že je odolný vůči zpoždění agentů až o $k$ časových úseků (kroků).

Plány označované jako $k$-robustní jsou vhodné v případě, kdy je předpokládané zpoždění všech agentů přibližně stejné a nezávislé na délce plánu a stavu prostředí, ve kterém se agent pohybuje. Využít je lze také v situaci, kdy nemáme představu o pravděpodobnosti, s jakou může ke zpoždění docházet.

## 2.2 *P*-robustnost

Druhý způsob zajištění robustnosti plánů, který byl představen v článku Atzmon a kol. (2019), má název $p$-robustnost. Ten používá pomocný pojem potenciální konflikt.

**Definice 11** (potenciální konflikt)**.** *Plán $\pi$ má potenciální konflikt $C = \langle i, j, t\rangle$, pokud existuje $\Delta(C) \geq 0$ takové, že se dva agenti $i$ a $j$ nachází na stejné pozici v čase $t$ a $t + \Delta(C)$. Formálně zapsáno platí $\pi_i[t] = \pi_j[t + \Delta(C)]$.*

Potenciální konflikt nastane, pokud se agent $i$ zpozdí při provedení $t$ akcí o $d_i \geq \Delta(C)$ časových úseků a agent $j$ se zpozdí o právě $d_i - \Delta(C)$ časových úseků při provedení $t + \Delta(C)$ akcí. Kolize agentů pak nastane v čase $t + d_i$.

Pravděpodobnost, že agent neprovede v daný časový úsek akci *move*, kterou má dle plánu udělat (zpozdí se), označme $p_d$. Nechť $P_0(\pi)$ značí pravděpodobnost, že se v plánu $\pi$, který počítá s konstantní pravděpodobností zpoždění všech agentů $p_d$, neobjeví žádný potenciální konflikt.

**Definice 12** ($p$-robustnost)**.** *Řekneme, že plán $\pi$ je **$p$-robustní**, pokud platí nerovnost $P_0(\pi) \geq p$.*

Výhodou $p$-robustních plánů je řešení právě těch konfliktů, které se jeví jako pravděpodobné. To je rozdíl oproti $k$-robustnosti, která řeší všechny konflikty pro nějakou pevnou hodnotu $k$. Známe-li dostatečné informace o prostředí a spolehlivosti agentů, je tato cesta snazší pro eliminaci kolizí díky přesnému zacílení v daném prostředí.

Na druhou stranu jsou ale znalosti o pravděpodobnosti zpoždění nutné pro samotnou konstrukci plánu, a pokud je neodhadneme dobře, nebudou výsledné plány příliš kvalitní. Nedostatkem může být i značná časová náročnost tohoto přístupu.

# 3. Způsoby řešení MAPF

V této kapitole se seznámíme s možnostmi, jak můžeme MAPF problém řešit. Poskytneme základní rozdělení do několika skupin a uvedeme konkrétní příklady algoritmů. Následně se podrobněji zaměříme na ty algoritmy a techniky, s nimiž se v práci dále setkáme.

Naším obecným cílem je najít nekolizní cesty z bodu $A$ do bodu $B$ pro skupinu $k > 1$ agentů. Samotné MAPF problémy lze rozdělit do dvou skupin, a sice na *centralizované* a *distribuované*. V centralizovaném přístupu máme jen jeden zdroj výpočetního výkonu (plánovač), který rozvrhuje cesty všech agentů sám a během jejich konstrukce má přehled o všech cestách současně. Plánovač dále zajišťuje garanci toho, že řešení je validní.

V distribuovaném přístupu máme zpravidla plně autonomní agenty s vlastním výpočetním výkonem a každý z nich se stará o nalezení vlastní cesty sám. Díky této masivní paralelizaci a dostatku výkonu dosahujeme zpravidla rychlejšího vyřešení MAPF problému (Pianpak a kol., 2019). Aby se zabránilo konfliktům, musí mezi sebou agenti navzájem komunikovat a svoje cesty postupně modifikovat. Navíc nemůžeme počítat s optimalitou a často ani s úplností.

I v případě centralizovaného přístupu můžeme při dostatku výpočetního výkonu hledat jednotlivé cesty paralelně, základním rozdílem je ale schopnost plánovače vidět cesty všech agentů najednou. V této práci řešíme MAPF problémy centralizovaným přístupem a dále se budeme primárně věnovat jemu.

Nyní přejděme k možnostem, jak (centralizovaný) MAPF problém řešit. Na úvod zmiňme, že nalezení optimálního řešení obecného MAPF je NP-těžké (Surynek, 2010). Proto jsme někdy z časových důvodů nuceni použít techniky vedoucí k nalezení řešení, které nemají tak dobré vlastnosti, jaké bychom si přáli. U řešení MAPF sledujeme následující dvě vlastnosti.

- **Optimalita** – nalezené řešení je *optimální*, pokud má nejlepší možnou hodnotu účelové funkce ze všech řešení. V opačném případě mluvíme o *suboptimálním* řešení.
- **Úplnost** – algoritmus nazveme *úplný*, pokud najde řešení vždy, když existuje. Jinak ho nazveme *neúplný*.

## 3.1 Řešení pomocí prohledávání

Jak již bylo uvedeno, chceme v úlohách MAPF najít cesty z bodu $A$ do bodu $B$ pro větší množství agentů. Typickým přístupem k řešení problému, ve kterém hledáme cesty, je prohledávání stavového prostoru. Způsob, jakým budeme stavový prostor prohledávat, pak dále ovlivní optimalitu i úplnost řešení.

1. **Neúplné a suboptimální algoritmy** – jedná se o rychlé algoritmy, které se snaží o hledání cest jednotlivých agentů nezávisle s následným vypořádáním se s konflikty. Patří sem např. Cooperative A* a WCHA* (Silver, 2005).
2. **Úplné a suboptimální algoritmy** – používá se řada algoritmů, které poskytují úplnost pro určité třídy grafů (TASS (Khorshid a kol., 2011), Push

and Swap (Luna a Bekris, 2011)). Již dříve byl představen polynomiální algoritmus, který je úplný pro libovolný graf (Kornhauser a kol., 1984).

3. **Úplné a optimální algoritmy** – první možností, jak zajistit úplnost a optimalitu, je prohledávání stavového prostoru, který v sobě zahrnuje kombinace stavů všech jednotlivých agentů. Mezi zástupce můžeme zařadit algoritmy A* (Hart a kol., 1968) (upravený pro hledání cest více agentů současně) a jeho vylepšení M* (Wagner a Choset, 2011).

   Dále existuje druhá skupina algoritmů, založená na dvouúrovňovém prohledávání. Zjednodušeně lze říci, že na vyšší úrovni se systematicky generují podmínky na kandidáty řešení tak, aby byla garantována optimalita a případně se zde kontrolují i konkrétní příklady řešení, které přišly z nižší úrovně. Na nižší úrovni se snažíme pomocí prohledávacího algoritmu najít jednotlivé cesty pro každého z agentů za podmínek, které byly nastaveny na vyšší úrovni. Řadíme sem techniky ITCS (Sharon a kol., 2011) a CBS (Sharon a kol., 2015).

### 3.1.1 Algoritmus A* a jeho rozšíření pro více agentů

Jedním z nejznámějších algoritmů na poli umělé inteligenci je nepochybně algoritmus A*. Tento algoritmus, používaný pro hledání nejkratších cest, můžeme snadno rozšířit pro nalezení optimálního a úplného řešení MAPF.

Pro skupinu $k$ agentů bude počátečním stavem množina $k$ startovních vrcholů a koncovým stavem množina $k$ cílových vrcholů. Přechod z jednoho stavu do druhého je dán provedením právě jedné akce každého z agentů. V takto definovaném stavovém prostoru lze pomocí algoritmu A* najít nejkratší cestu, která bude plánem s nejnižší možnou cenou.

Abychom mohli zaručit, že nalezená cesta neobsahuje konflikty, musíme je eliminovat při generování následníků daných stavů. Zamezení vrcholového konfliktu lze snadno kontrolovat pouze na základě znalosti tohoto stavu. Pro zamezení konfliktů vzájemné výměny vrcholů v plánu potřebujeme kontrolovat dvojice stavů $(rodič, potomek)$. A pokud bychom chtěli tímto způsobem implementovat $k$-robustní algoritmus, musíme posuzovat konflikty pro každou $k$-tici po sobě jdoucích stavů.

Asi největší nevýhodou zmíněného přístupu je počet vygenerovaných stavů. Ačkoliv v průběhu výpočtu určité množství stavů zamítneme na základě výskytu konfliktů, narážíme na vysoký větvící faktor. Pro $k$ agentů a $n$ možných akcí generuje jeden stav až $n^k$ nových stavů.

Vhodnost použití algoritmu tak velmi rychle klesá s počtem agentů v MAPF problému. Ideálně bychom ho chtěli použít spíše pro hledání jednotlivých cest na nižší úrovni, což se děje v rámci algoritmu CBS.

### 3.1.2 Conflict based search (CBS)

Jedná se o algoritmus garantující nalezení optimálního řešení, který pracuje ve dvou úrovních. Na nižší úrovni hledáme nejkratší cestu pomocí prohledávání (použijeme algoritmus A*) vždy pro jednoho agenta, čímž značně omezíme stavový prostor oproti algoritmu v podkapitole 3.1.1. Na vyšší úrovni hledáme optimální

plán ve stromu omezujících podmínek *(constraint tree)*. Jedná se o binární strom, kde každý uzel $N$ obsahuje tři důležité položky:

- **Omezení** – množina trojic $\langle i, v, t\rangle$, která říká, že se agent $i$ nesmí nacházet v časovém okamžiku $t$ ve vrcholu $v$ (pro zamezení vrcholových konfliktů) a množina čtveřic $\langle i, v_1, v_2, t\rangle$, zakazující přejezd agenta $i$ z vrcholu $v_1$ do vrcholu $v_2$ v časovém intervalu od $t$ do $t+1$ (pro zamezení konfliktů vzájemné výměny vrcholů). Každý uzel přebírá všechna omezení svého rodiče.
- **Řešení** – množina cest pro jednotlivé agenty, která je konzistentní s omezeními v uzlu.
- **Celkovou cenu** - popisuje kvalitu nalezené množiny řešení. Aby bylo možné garantovat optimalitu nalezeného řešení, musí hodnota ceny uzlu korespondovat s hodnotou účelové funkce řešeného MAPF problému.

Začínáme s počátečním uzlem (kořenem), který obsahuje prázdnou množinu omezení. Řešení v uzlu tvoří cesty, které pro jednotlivé agenty nalezl algoritmus na nižší úrovni. Cena je nastavena dle kvality řešení. Cílovým uzlem nazveme uzel, jehož řešení neobsahuje žádné konflikty.

Dokud nemáme cílový uzel, vybereme list stromu s nejnižší hodnotou celkové ceny. Následně zkontrolujeme, zda nějaká dvojice cest obsahuje konflikty. Pokud ne, prohlásíme uzel za cílový a algoritmus skončí. V opačném případě existuje v řešení vrcholový konflikt, který můžeme zapsat ve tvaru $\langle i, j, v, t\rangle$, popisující, že agenti $i$ a $j$ mají konflikt ve vrcholu $v$ v čase $t$. Nebo existuje konflikt vzájemné výměny vrcholů tvaru $\langle i, j, v_1, v_2, t\rangle$, který nám říká, že agenti $i$ a $j$ v časech $t$ a $t + 1$ navzájem přejeli z vrcholů $v_1$ do $v_2$ a naopak.

Každý validní plán pak musí vzhledem k povaze konfliktu splnit jednu ze dvou podmínek. V případě vrcholového konfliktu se buď v čase $t$ nesmí ve vrcholu $v$ nacházet agent $i$, nebo se tam nesmí nacházet agent $j$. Pro konflikt vzájemné výměny vrcholů pak buď agent $i$ nesmí v čase $t$ začít přejíždět z $v_1$ do $v_2$, nebo agent $j$ nesmí přejíždět z $v_2$ do $v_1$.

K aktuálnímu uzlu s konfliktem vytvoříme dva potomky. Do jednoho z nich přidáme omezení tvaru $\langle i, v, t\rangle$, resp. $\langle i, v_1, v_2, t\rangle$ a do druhého z nich omezení $\langle j, v, t\rangle$, resp. $\langle j, v_2, v_1, t\rangle$ pro vrcholový konflikt, resp. konflikt vzájemné výměny vrcholů. Předchozí množinu řešení pak musíme upravit nalezením nové cesty pro agenta, ke kterému se vztahuje omezení a to tak, aby bylo řešení opět konzistentní. Následně pokračujeme výběrem dalšího listu stromu a celý proces opakujeme až do nalezení uzlu, jehož řešení je validní.

### Přidání k-robustnosti

Základní úprava algoritmu CBS pro hledání $k$-robustních plánů je jednoduchá a přímočará. Pokud dojde při validaci řešení k nalezení $k$-zpožděného konfliktu $c = \langle i, j, t\rangle$, vytvoříme aktuálnímu uzlu dva potomky, přičemž do prvního z nich přidáme omezení tvaru $\langle i, v, t\rangle$ a do druhého omezení tvaru $\langle j, v, t + \Delta\rangle$, kde $v$ značí vrchol, ve kterém ke konfliktu došlo.

Z důvodu zvýšení rychlosti se ale častěji setkáváme s vylepšeným $k$-robustním CBS *(Improved kR-CBS)* (Atzmon a kol., 2018). Od základní úpravy se liší pouze tím, že v uzlech stromu omezujících podmínek používá tzv. *intervalová omezení*

*(range constraint)*. Intervalové omezení je trojice $\langle i, v, [t_1, t_2]\rangle$ udávající, že agent $i$ se ve vrcholu $v$ nesmí nacházet v žádném časovém okamžiku z intervalu $\langle t_1, t_2\rangle$. V praxi pak používáme symetrickou variantu omezení a pro $k$-zpožděný konflikt vytváříme z daného uzlu dva potomky s novými intervalovými omezeními $\langle i, v, [t, t+k]\rangle$ a $\langle j, v, [t, t+k]\rangle$. Atzmon a kol. (2018) dokázali, že i takto upravený algoritmus je pro $k$-robustnost optimální a úplný.

#### Pseudokód algoritmu

Pro úplnost ještě níže uvádíme pseudokód základní verze algoritmu CBS (zdroj Sharon a kol. (2015)).

```
                    Algoritmus CBS
Vstup: instance MAPF problému s k agenty
kořen.Omezení = null
kořen.Řešení = najdi nejkratší cesty pro jednotlivé agenty
kořen.Cena = makespan(kořen.Řešení)
vlož kořen do haldy OPEN
while OPEN neprázdná:
   P <-- vyber uzel z OPEN         //má nejnižší makespan
   zkontroluj konflikty v P
   if P nemá konflikt:
      return P.Řešení
   C <-- konflikt v P tvaru (i,j,v,t)   //nebo (i,j,v1,v2,t)
   foreach agent ag in C:     //tedy postupne pro agenty i, j
      A <-- nový uzel
      A.Omezení = P.Omezení + (ag,v,t)
      A.Řešení = P.Řešení
      najdi cestu pro ag v souladu s A.Omezení a uprav A.Řešení
      A.Cena = makespan(A.Řešení)
      if A má řešení
         přidej A do OPEN
```

## 3.2 Převod na jiný problém

Protože MAPF patří do třídy NP-těžkých problémů, můžeme ho zkusit převést na jiný problém z této třídy, který následně vyřešíme použitím vhodného existujícího řešiče. Převádět lze například na problém splnitelnosti (SAT), splňování omezujících podmínek (CSP) nebo lineární programování. Výhodou převodu je existence výkonných optimalizovaných řešičů pro tyto problémy, které nyní můžeme použít v MAPF. Jedná se tedy o pěkný příklad využití znalostí z jiných oborů pro řešení našeho problému.

### 3.2.1 Převod MAPF na problém splňování omezujících podmínek

V této práci používáme převod MAPF problému na deklarativní model, založený na splňování omezujících podmínek. Pro implementaci se hodí programovací

jazyk Picat (Zhou a kol., 2015). Při konstrukci modelu jsme vycházeli z postupu uvedeného v článku Barták a kol. (2017).

Principem deklarativního popisu problému je definice proměnných a vyjádření podmínek, které musí validní řešení MAPF splňovat. Řešič pak nalezne přípustné ohodnocení všech proměnných z modelu, čímž také najde řešení MAPF problému. Vzhledem k tomu, že v MAPF přesně nevíme, jak dlouhá je délka výsledného plánu, ale pro zakódování potřebujeme definovat předem daný počet proměnných, přikročíme k následující úpravě. Při převodu budeme kódovat vždy plány pevně dané délky s tím, že pokud řešení dané délky neexistuje, zvýšíme ji v dalším kroku o jedna. Začínáme vždy od minimální možné délky plánu, abychom měli jistotu, že první vyhovující řešení má nejlepší možnou délku z hlediska účelové funkce. Dodejme, že v tomto typu MAPF problémů se běžně používá funkce makespan.

Základní myšlenka převodu je následující. Použijeme vrstevnatý graf popisující údaj o poloze agenta v čase. Zavedeme booleovské proměnné $B_{t,a,v}$ říkající, zda se agent $a$ nachází v čase $t$ ve vrcholu $v$. Níže uvádíme příklad zápisu omezujících podmínek pro MAPF problém (pro jednoduchost jen s omezením vrcholových konfliktů). Podmínky jsou uvedeny vždy nejprve slovně a následně pomocí zapisu v jazyce Picat.

1. Každý agent musí být v každém časovém úseku v právě jednom vrcholu grafu.

   $\sum_v B_{t,a,v} = 1$, pro všechny časové úseky $t$ a všechny agenty $a$.

2. Žádní dva agenti nejsou ve stejný čas ve stejném vrcholu (nenastává vrcholový konflikt).

   $\sum_a B_{t,a,v} \leq 1$, pro všechny časové úseky $t$ a vrcholy $v$.

3. Z daného vrcholu $v$ může agent $a$ přejet v následujícím časovém úseku $t + 1$ do libovolného vrcholu, který je sousedem vrcholu $v$ (a to včetně $v$ samotného).

   $B_{t,a,v} = 1 \Rightarrow \sum_{u \in sousedé(v)} B_{t+1,a,u} \geq 1$, pro všechny vrcholy $v$, agenty $a$ a časové úseky $t$ (kromě posledního).

*Poznámka.* V podmínce 3 si můžeme ve výrazu $\sum_{u \in sousedé(v)} B_{t+1,a,u} \geq 1$ dovolit napsat nerovnost, protože rovnost vynutí podmínka 1. Barták a kol. (2017) ve svém článku uvádějí, že užití nerovnosti v podmínce 3 je při převodu do konjunktivní normální formy více efektivní.

### Přidání k-robustnosti

Výše popsaný model umožňuje velmi jednoduše přidat podmínku, která zajistí garanci $k$-robustnosti řešení. Touto podmínkou nahrazujeme omezení zakazující výskyt vrcholového konfliktu.

4. Žádní dva agenti nejsou ve stejném vrcholu v rozmezí $r$ časových úseků (robustnost zde určuje parametr $r$).

   $B_{t,a,v} = 1 \Rightarrow \sum_{prev \in \langle t-r,t \rangle, a_2 \neq a} B_{prev,a_2,v} = 0$, pro každý časový úsek $t$, vrchol $v$ a každého agenta $a$.

# 4. Robustnost zajištěná alternativními plány

V této kapitole představíme nový přístup hledání robustních cest pomocí plánování s alternativami.

## 4.1 MAPF v nedeterministickém prostředí

Jak jsme si již ukázali v předchozí kapitole, existují rozličné způsoby hledání klasického řešení MAPF problému. Výstupem pro každého agenta je něco, čemu říkáme plán. Plán obsahuje cestu zakódovanou pomocí vrcholů. Můžeme se na něj ale také dívat jako na posloupnost akcí (*move* nebo *wait*), které agenta provedou z počáteční pozice do cíle.

Plánovač hledající vyhovující řešení pro všechny agenty nepočítá s přítomností nejistoty ve světě, kde se tito agenti pohybují. Konstrukce řešení probíhá na základě předpokladu, že vykonání akcí proběhne zcela deterministicky. Ve chvíli, kdy do MAPF problému zavedeme pojem zpoždění, vnášíme do celého systému určitou míru nejistoty a nedeterminismu. Nyní už není dopředu zřejmé, zda se konkrétní agent v daném čase opravdu posune z bodu $A$ do bodu $B$, nebo jestli zůstane stát na místě. Konstrukcí robustních plánů se snažíme na tyto věci dopředu připravit.

Dříve zmíněná technika $k$-robustnosti předem odhadne, jak by mohl vypadat nejhorší možný případ a pro ten pak hledá odpovídající plán. $P$-robustnost se vypořádává s nepřízní osudu tím, že odhaduje pravděpodobnosti neočekávaných událostí. Jako řešení vrátí plán, který pro dané pravděpodobnosti má statisticky vysokou šanci uspět. My nyní představíme nový koncept založený na plánování s alternativami.

## 4.2 Plánování s alternativami

Zajímavou technikou, kterou používáme v plánování v nedeterministickém prostředí s možnými přechody do více stavů, je plánování s alternativami (v originále *contingency planning*), popsané např. v knize Russell a Norvig (2009) nebo článku Pryor a Collins (1996). Místo posloupnosti stavů/akcí pracujeme se stromem. V kořeni tohoto stromu máme počáteční stav systému. Každý další uzel pak obsahuje pro všechny možné předpokládané následníky větev vedoucí do stavu, který jsme si označili jako cílový. Výběr konkrétní větve ve stromu plánů probíhá až za běhu. Rozhodnutí činíme na základě získaných informací z vnějšího prostředí.

Popsaný způsob přípravy plánu zároveň dobře odpovídá tomu, jak se lidé vypořádávají s neurčitostí v každodenním životě. Mají nějaký plán, kterého se snaží držet, a zároveň si připraví jisté záložní plány pro případ, kdy se něco konkrétního pokazí. Z počátku je rozumné předpokládat, že vše půjde „normálně“ a teprve ve chvíli, kdy je klidový průběh ohrožen, přejdeme na záložní plán. Evakuační plány pro případ požáru najdeme v každé větší budově, podniky v době krize postupují

dle plánu krizového řízení a státy mají například připravené postupy v případě různých pandemií.

Plánování s alternativami využijeme pro konstrukci řešení MAPF, které tím bude do jisté míry chráněné vůči kolizím. Výskyt zpoždění v plánu lze chápat jako neočekávanou akci, která naruší normální průběh předem spočítaného deterministického plánu a zavede nás na cestu, která s touto situací počítá a bezpečně ji řeší.

## 4.3 Zavedení robustnosti pomocí plánování s alternativami

Pro daný MAPF problém již na začátku připravíme více možných scénářů toho, jak se může vykonávání plánů vyvíjet a konkrétní podoba projeté cesty bude vytvořena na základě znalostí z okolního světa v průběhu provádění plánu. Potřebné informace nám budou poskytovat senzory vnímající okolní prostředí. Místo plánů pro jednotlivé agenty tak, jak jsme je znali doposud, zavedeme nový pojem plánu s alternativami.

**Definice 13** (plán s alternativami)**.** *Pro agenta $i$ je* ***plán s alternativami*** $\pi_i$ *dvojice* $\langle \pi_i^h, \mathcal{A}_i \rangle$*, kde:*

- $\boldsymbol{\pi_i^h}$ *je plán pro jednoho agenta (dle definice 2), který označíme jako hlavní,*
- $\boldsymbol{\mathcal{A}_i}$ *je množina alternativ obsahující trojice tvaru* $\langle k, \langle min, max \rangle, \pi_i^a \rangle$*, kde:*
  - $\boldsymbol{k}$ *je krok agenta v hlavním plánu,*
  - $\boldsymbol{\langle min, max \rangle}$ *je interval výběru udávající velikost zpoždění oproti $k$, při kterém se vybere alternativní plán $\pi_i^a$ (min, max* $\in \mathbb{N}$*),*
  - $\boldsymbol{\pi_i^a}$ *je alternativní plán (alternativní cesta), pro který platí, že:*
    - *startovní vrchol alternativního plánu $s_a(i) = \pi_i^h[k]$,*
    - *cílový vrchol alternativního plánu $t_a(i) = t(i)$.*

*Poznámka* (k výběru alternativního plánu)*.* Agent $i$ si vybere alternativní plán $\pi_i^a$ z vrcholu $\pi_i^h[k]$, pokud provedl právě prvních $k$ akcí z hlavního plánu za $z + k$ kroků, kde $z$ patří do intervalu výběru příslušné alternativy.

**Definice 14** (minimální zpoždění)**.** *Minimálním zpožděním pro výběr alternativního plánu rozumíme dolní mez intervalu výběru z příslušné alternativy.*

V MAPF problému s $m$ agenty tvoří množina $\pi^h = \{\pi_1^h,...,\pi_m^h\}$ *hlavní řešení* tohoto problému. Od hlavního řešení budeme vždy požadovat, aby bylo validní (ve smyslu definice 7) a kromě toho také optimální vzhledem k účelové funkci.

V našem případě budeme vždy začínat získáním hlavního řešení. Následně k němu připojíme alternativní plány v místech, kde se budeme snažit vyhnout pravděpodobnému riziku kolize. Tyto plány chápeme jako cesty vedoucí z nějakého vrcholu hlavního plánu do cílové pozice agenta, jejichž délka ani průběh nejsou pevně dané. Počet alternativ, které jsou k dispozici pro agenta $i$ udává velikost množiny $\mathcal{A}_i$. Nastane-li během vykonávání plánu předem dané zpoždění, odchýlí se agent na tomto místě od hlavního plánu k alternativnímu.

Při volbě alternativy předpokládáme, že intervaly výběru v jednom vrcholu jsou navzájem disjunktní a vybrání lze provést jednoznačně. Z definice plyne, že jeden alternativní plán lze vybrat pro více velikostí zpoždění. Po všech plánech s alternativami navíc budeme požadovat, aby jejich alternativní plány byly validní.

**Definice 15** (validní alternativní plán)**.** *Alternativní plán $\pi_i^a$ agenta $i$ je validní, pokud při žádné velikosti zpoždění vedoucí k jeho vybrání nenastane konflikt s hlavními plány ostatních agentů v řešení MAPF, jehož je alternativní plán součástí.*

*Příklad.* Nechť $\pi = \{\langle \pi_1^h, \mathcal{A}_1 \rangle, \langle \pi_2^h, \mathcal{A}_2 \rangle\}$ je nějaké řešení MAPF problému se dvěma agenty, které obsahuje plány s alternativami. Dále uvažme, že:

- $\pi_1^h$ obsahuje plán určený posloupností vrcholů $s(1) - v_1 - v_2 - t(1)$,
- $\pi_2^h$ obsahuje plán určený posloupností vrcholů $s(2) - v_3 - v_4 - v_5 - t(2)$,
- $\mathcal{A}_1 = \{\langle 1, \langle 1, 2\rangle, \pi_1^{a_1}\rangle, \langle 2, \langle 3, 3\rangle, \pi_1^{a_2}\rangle\}$, kde:
  - $\pi_1^{a_1}$ je cesta obsahující vrcholy $v_1 - v_5 - v_6 - t(1)$,
  - $\pi_1^{a_2}$ je cesta obsahující vrcholy $v_2 - v_3 - v_7 - t(1)$,
- $\mathcal{A}_2 = \emptyset$.

Potom alternativní plán $\pi_1^{a_2}$ je validní, zatímco plán $\pi_1^{a_1}$ validní není. Vybereme-li si totiž cestu $\pi_1^{a_1}$ při zpoždění velikosti 1, nastane vrcholový konflikt s plánem $\pi_2^h$ ve vrcholu $v_5$ v čase 3.

Pokud bychom odstranili číslo 1 z intervalu výběru první alternativy (takže nový interval by byl tvaru $\langle 2, 2\rangle$), stane se alternativní plán $\pi_1^{a_1}$ také validním.

Každý alternativní plán je tedy robustní vůči určité množině zpoždění a chrání ostatní agenty před kolizí s agentem, který se opozdil. Konkrétní implementace variant robustnosti založených na technice *contingency planning*, resp. alternativních plánech ukážeme později. Nyní ještě vyřešme několik obecných otázek.

### 4.3.1 Jak vybírat alternativní plány

Ke změně aktuální větve a přesunu z hlavního plánu na alternativní můžeme přistupovat dvěma způsoby. V případě centralizovaného přístupu bude o výběru rozhodovat plánovač. Ten daného agenta vždy pouze informuje, kterou cestou se má vydat a agent zvolený plán slepě vykoná. Do jisté míry se tak už jedná spíše o on-line přeplánování, kterému se detailně věnuje např. článek (Atzmon a kol., 2020). Zde jen uvedeme, že výhodou popsaného přístupu je možnost plánovače vidět všechny agenty najednou a volit alternativní plány ve shodě s polohou každého z nich. Nevýhodou centrálního přístupu je nutnost rychlé a bezchybné komunikace plánovače s agenty a to oběma směry.

Druhým způsobem implementace plánů s alternativami do MAPF je distribuovaný přístup. Zde přesuneme rozhodnutí o změně plánu na samotného agenta. Ve chvíli, kdy se objeví ve vrcholu, ze kterého se dá vydat na další cestu více možnostmi, zvolí si na základě informací ze svých senzorů vhodnou alternativu.

Celý proces proběhne nezávisle na ostatních agentech. Odpadá tedy nutnost komunikace s plánovačem, což je hlavní výhodou tohoto přístupu. Na druhou stranu vyvstává problém vznikající právě kvůli vzájemné nezávislosti rozhodnutí jednotlivých agentů. Žádný z nich totiž neví nic o tom, jaké alternativy zvolili ve svých plánech ostatní agenti a kde se právě teď pohybují. Musíme tak řešit hrozbu konfliktů mezi hlavními a alternativními plány, případně i mezi dvěma alternativními plány navzájem.

Nakonec připomeňme, že ačkoliv se jedná o distribuovaný způsob výběru alternativy, pohybujeme se stále v centralizovaném způsobu hledání řešení, protože konstrukci alternativ provádí plánovač.

Jiným problémem, který musíme řešit, je rostoucí počet možných cest při větvení. Čím více alternativ v plánech dovolíme (zvýšíme odolnost vůči kolizi), tím více času spotřebujeme na plánování. Myšlenka přístupu *contingency planning* spočívala v přidání akce pro každý možný senzorický vstup z prostředí do každého místa plánu. Z hlediska MAPF se jeví tento postup jako časově neúnosný a také neefektivní, protože každý výskyt zpoždění není nutné ihned řešit. Množství alternativních cest, které budeme vytvářet, bude nutné rozumně omezit.

Vzhledem k tomu, jak jsme navrhli definici plánu s alternativami, máme jistotu, že se odbočky od hlavního plánu už dále větvit nebudou. Díky zvolenému omezení jsme tak zabránili exponenciálnímu nárůstu možných plánů, které teoreticky mohou během vykonávání agenti provést. S tím je samozřejmě spojeno menší množství kontrol a podmínek na bezkolizní průběh těchto cest. Navíc si každý agent během provádění plánu zvolí nejvýše jednu alternativu, kterou se následně vydá do cíle.

Z uvedeného vyplývá, že při popisu konkrétní implementace robustnosti založené na plánech s alternativami nás zajímají především dvě věci. Za prvé, jakým způsobem a do jaké míry chceme řešit konflikty v alternativních plánech agentů navzájem. Za druhé, jak omezíme počty alternativ, které konstruujeme.

Pojďme se nyní zabývat způsobem výběru vhodných míst v plánech, ve kterých bychom umožnili větvení.

### 4.3.2 Kde hrozí kolize v plánech

K alternativnímu plánu se budeme uchylovat v případě, kdy by následování hlavního plánu způsobilo nějakou katastrofu, v našem případě kolizi dvou agentů. Nevadí nám tak úplně samotný výskyt zpoždění, jako spíš to, co někdy v budoucnu způsobí jeho následky. Máme-li agenta, který se někde neočekávaně opozdí, mohou v dalším průběhu plánu nastat dva možné scénáře.

1. Agent se opozdí, ale tato anomálie nebude mít žádný vliv na ostatní agenty a projeví se pouze zvýšením délky cesty. Jediným důsledkem může být zvětšení hodnoty účelové funkce, kterou se snažíme minimalizovat. U funkce SoC se toto projeví vždy, u makespanu opoždění může, ale i nemusí hrát roli. Varianta nevyžaduje nutnost změny plánu.

2. Agent se opozdí a následkem toho se v některém z dalším vrcholů v plánu srazí s jiným agentem. Tato varianta vyžaduje nutnost změny plánu.

Můžeme nahlédnout, že místa potenciálních srážek jsou známá již po obdržení konkrétního (hlavního) řešení. Patří sem právě ty vrcholy, ve kterých se kříží

plány alespoň dvou agentů. Tuto množinu vrcholů pro plán $\pi$ nazveme množinou konfliktních vrcholů.

**Definice 16** (množina konfliktních vrcholů)**.** *Pro hlavní řešení $\pi = \{\pi_1^h, ..., \pi_n^h\}$ nazveme množinu $C_\pi = \{v \in V : \exists i,j,x,y : i \neq j \wedge \pi_i[x] = \pi_j[y] = v\}$ množinou konfliktních vrcholů.*

Jako ideální se jeví budovat alternativní cesty za účelem odklonu agentů před průjezdem vrcholy, které patří do výše zmíněné množiny. Ne každý vrchol z množiny konfliktních vrcholů nicméně představuje stejné riziko. Zde musíme rovněž vzít v potaz údaj, který nazveme hlavní bezpečný interval.

**Definice 17** ((hlavní) bezpečný interval agenta ve vrcholu)**.** *Pro (hlavní) řešení $\pi = \{\pi_1,...,\pi_n\}$, agenta $i$ a vrchol $v \in \pi_i$, nazveme (hlavním) bezpečným intervalem časové rozmezí, ve kterém může agent $i$ vrcholem $v$ projet, aniž by se musel obávat kolize s některým dalším agentem jedoucím po cestě dané svým (hlavním) plánem.*

V kontextu této kapitoly použijeme definici ve významu hlavní bezpečný interval, svůj smysl však má i pro klasické řešení MAPF, jak ukážeme později.

Délkou hlavního bezpečného intervalu rozumíme velikost časového rozmezí. Má-li agent $i$ projet vrcholem $v$ v čase $t_1$, pak dolní mezí hlavního bezpečného časového intervalu vrcholu $v$ bude číslo $t_1$. Dříve se daný agent do vrcholu určitě nedostane. Pokud vrchol $v$ náleží do množiny $C_\pi$, pak horní mezí bude číslo $t_2 - 1$, kde $t_2$ udává první příjezd dalšího agenta do vrcholu po agentovi $i$. Pokud žádný takový není, pak je horní mez teoreticky nekonečno, v praxi zpravidla chceme omezit délku hlavních bezpečných intervalů vhodnou konstantou $k$. Z popisu hlavního bezpečného intervalu plyne pro exekuci plánu $\pi$ s $k$ agenty následující tvrzení.

**Tvrzení 2.** *Pobývá-li každý z agentů ve vrcholech hlavního plánu pouze v časech, které patří do jeho hlavního bezpečného intervalu, pak v plánu $\pi$ nenastane kolize.*

*Důkaz.* Hlavní bezpečné intervaly žádných dvou agentů se z definice nepřekrývají.

□

Pokud při provádění plánu dojde vinou zpoždění k tomu, že agent $i$ projede vrcholem $v$ mimo svůj hlavní bezpečný interval, hovoříme o jeho *přesažení*. Přesažením intervalu ztrácíme kontrolu nad bezkolizním provedením plánu, nicméně v danou chvíli už to není možné nijak zvrátit. Průběžným sledováním velikosti zpoždění však můžeme hrozbu přesažení intervalu ve vrcholu detekovat dříve v některém z jeho předchůdců.

**Definice 18** (hrozba $k$-konfliktu)**.** *Agent detekuje hrozbu $k$-konfliktu ve vrcholu $u$, pokud by při aktuálním zpoždění $v$ tomto vrcholu, čítajícím nejvýše $k$ kroků, nastalo přesažení bezpečného intervalu v některém z dalších vrcholů jeho hlavního plánu.*

Hrozbu $k$-konfliktu se budeme snažit odstranit zvolením alternativy. Jestliže agent vybere při detekci hrozby $k$-konfliktu alternativní plán, označíme tuto

hrozbu jako *eliminovanou*. Konkrétní hrozba se vztahuje vždy k vrcholu, ve kterém by došlo k přesažení. Ačkoliv ji lze v určitých případech detekovat při průjezdu několika vrcholy za sebou, eliminaci potřebujeme provést jen jednou.

Dále představíme dvě konkrétní metody robustnosti založené na plánech s alternativami.

## 4.4 Alternativní *k*-robustnost

Začneme s popisem robustního plánu, který vychází z dříve definovaného pojmu $k$-robustnosti. Ještě předtím ale definujeme jeden pomocný pojem.

**Definice 19** ($k$-robustní eliminace)**.** *Nechť je dán MAPF problém a jeho řešení. Hrozbu k-konfliktu eliminujeme k-robustně, pokud je alternativní plán $\pi_i^a$ agenta i vybraný při minimálním zpoždění k-robustní vůči hlavním plánům ostatních agentů.*

Návaznost na minimální zpoždění má dobrý smysl, protože udává velikost zpoždění, kterou bychom při detekci $k$-konfliktu očekávali asi nejčastěji. Navíc v případě, kdy by agent pokračoval s minimálním zpožděním ve vykonávání hlavního plánu, došlo by při přesažení bezpečného intervalu ke kolizi s jiným agentem, pokud by tento agent jel na čas.

Při $k$-robustní eliminaci konfliktu je dle předchozí definice vybraný alternativní plán validní až pro zpoždění o $min + k$ kroků, kde $min$ udává dolní mez intervalu výběru. Nyní už přejděme k samotné definici robustnosti.

**Definice 20** (alternativní $k$-robustnost)**.** *Řešení MAPF problému $\pi$ obsahující plány s alternativami $\pi_1, ..., \pi_n$ nazveme **alternativně k-robustní**, pokud jsou k-robustně eliminovány všechny k-konflikty v nejbližších předchůdcích nepatřících do množiny konfliktních vrcholů tohoto řešení.*

### 4.4.1 Předpoklady

Předpokládáme agenty, kteří disponují určitými druhy senzorů. Využijeme senzor zjišťující současnou polohu (GPS) a senzor měřící aktuální čas. Také nastavíme omezení na maximální délku hlavního bezpečného intervalu na hodnotou $k$. Při zpoždění agentů o více než $k$ kroků se už totiž ocitáme mimo garance alternativně $k$-robustního plánu, takže pro nás není příliš důležité.

### 4.4.2 Konstrukce plánu

Místo řešení $\pi$ skládajícího se z klasických cest jednotlivých agentů konstruujme řešení obsahující plány s alternativami. Nejdříve nalezneme řešení MAPF problému označované jako hlavní. Po jeho obdržení jsme v něm schopni nastavit každému agentovi v každém vrcholu hlavní bezpečný interval. Následně přejdeme ke konstrukci alternativ.

Nechť máme zadáno číslo $k$ označující alternativní $k$-robustnost plánu. Pokud mají všichni agenti ve vrcholech hlavní bezpečný interval délky $k$, jsme hotovi. Mimochodem, takový plán by byl $k$-robustní, protože bychom zde nenašli žádný

$k$-zpožděný konflikt (viz definice 9). Pokud podmínka splněna není, nechť $B$ je množina všech hlavních bezpečných intervalů s délkou menší než $k$.

Prvky z množiny $B$ začneme zpracovávat jeden po druhém. Podívejme se nyní blíže na jeden takový prvek. Ten se týká agenta $i$ ve vrcholu $v$ a udává, že délka hlavního bezpečného intervalu $d$ je menší, než bychom si přáli. Jinými slovy, vrcholem $v$ projede další agent po $d+1$ časových úsecích od předpokládaného projetí agenta $i$. Platí samozřejmě, že $d < k$.

Pro agenta $i$ připravíme alternativní plán, který $k$-robustně eliminuje hrozbu $k$-konfliktu. Plán povedeme z nejbližšího předchůdce vrcholu $v$ (označme ho jako $u$) nepatřícího do množiny konfliktních vrcholů. Pokud takový vrchol neexistuje, nemusíme se dle definice eliminací zabývat. Alternativní cestu pro tento prvek nekonstruujeme a pokračujeme na další.

Pokud vrchol $u$ máme, pak pro přidání alternativní cesty do hlavního plánu příslušného agenta použijeme prohledávací algoritmus A*. Hledat začneme ve vrcholu $u$ v čase $t_u+d+1$, kde $t_u$ značí plánovaný příjezd do vrcholu $u$. Následníkem vrcholu $u$ v algoritmu hledání cesty (a prvním uzlem alternativního plánu) bude pouze takový vrchol, který nemá žádnou část intervalu $\langle t_u+d+2, t_u+d+k+2\rangle$ vyhrazenou jako bezpečnou pro jiného agenta. Tím zajišťujeme $k$-robustnost pro dolní mez intervalu výběru plánu. Obecně řečeno, pro $n$-tý vrchol cesty bude platit, že časový interval $\langle t_u+d+n+1, t_u+d+k+n+1\rangle$ se nebude překrývat s hlavními bezpečnými intervaly ostatních agentů. Výše zkonstruovaný plán si agent $i$ dle definice vybere, pokud bude jeho zpoždění při průjezdu vrcholem $u$ v rozmezí od $d+1$ do $k+d+1$ kroků.

Vypořádáme-li se takto se všemi vrcholy z $B$, jsme hotovi. Pokud v nějakém bodě zjistíme, že cesta z vrcholu $u$ neexistuje, potom skončíme neúspěchem.

### 4.4.3 Striktní alternativní $k$-robustnost

Doposud jsme se podrobně věnovali analýze výběru míst pro alternativní plány. Nezabývali jsme se ale problémem vzájemné koordinace těchto plánů, co se konfliktů týče. Přirozeným požadavkem směrem k alternativním plánům by bylo zajištění jejich validity přinejmenším pro případ, který bychom očekávali jako nejpravděpodobnější. Formalizovat tento požadavek můžeme pomocí definice alternativního konfliktu.

**Definice 21** (alternativní konflikt)**.** *Řešení zahrnující plány s alternativami obsahuje alternativní konflikt, pokud v něm existují dva alternativní plány různých agentů a k nim příslušné hodnoty minimálního zpoždění takové, že při jejich vybrání nastane v plánech konflikt.*

*Příklad.* Nechť $\pi = \{\langle \pi_1^h, \mathcal{A}_1\rangle, \langle \pi_2^h, \mathcal{A}_2\rangle\}$ je nějaké řešení MAPF problému se dvěma agenty, které obsahuje plány s alternativami. Dále uvažme, že:

- $\pi_1^h$ obsahuje plán určený posloupností vrcholů $s(1) - v_1 - v_2 - t(1)$,
- $\pi_2^h$ obsahuje plán určený posloupností vrcholů $s(2) - v_3 - v_4 - v_5 - t(2)$,
- $\mathcal{A}_1 = \{\langle 1, \langle 2, 2\rangle, \pi_1^a\rangle\}$, kde:
  - $\pi_1^{a_1}$ je cesta obsahující vrcholy $v_1 - v_6 - v_8 - t(1)$,

- $\mathcal{A}_2 = \{\langle 2, \langle 1, 3\rangle, \pi_2^a\rangle\}$, kde:
  - $\pi_2^a$ je cesta obsahující vrcholy $v_4 - v_7 - v_8 - t(2)$,

Potom plány $\pi_1^a$, $\pi_2^a$ obsahují alternativní konflikt (ve vrcholu $v_8$ a v čase 5). Při zvýšení dolní meze intervalu výběru u alternativy v množině $\mathcal{A}_2$ na hodnotu 2 tento konflikt zmizí. Řešení už potom žádný jiný alternativní konflikt obsahovat nebude.

V ideálním případě bychom si přáli, aby naše řešení alternativní konflikt neobsahovalo. Výše popsaná konstrukce nicméně toto negarantuje, což nás přivádí k myšlence zesílit původní definici alternativní $k$-robustnosti.

**Definice 22** (striktní alternativní $k$-robustnost)**.** *Řešení MAPF problému nazveme* ***striktní alternativně k-robustní*** *, pokud je alternativně $k$-robustní a navíc neobsahuje žádný alternativní konflikt.*

Z čistě rozlišovacích důvodů budeme řešení, která jsou alternativně $k$-robustní, ale nevyžadujeme u nich podmínku striktnosti, označovat přídavným jménem *volná*.

### Konstrukce plánu

Oproti konstrukci ve volném přístupu přijdeme s jinou strategií hledání alternativ. Opět vytvoříme množinu $B$ a určíme místa, odkud chceme alternativní plány vést. Tentokrát je ale nebudeme hledat postupně, protože chceme, aby dohromady tvořily něco jako validní plán.

Potýkáme se s podobným problémem, který popisuje klasický MAPF problém a dostáváme nový, mírně upravený. Jestliže původní MAPF problém byl definovaný jako trojice $\langle G, s, t\rangle$, pak *upravený* MAPF problém definujeme trojicí $\langle G, s', t\rangle$, kde:

- $s' : A \longrightarrow 2^{V \times N}$ je funkce, která každému prvku (agentovi) z množiny $A$ přiřazuje množinu dvojic $(v, n)$ udávající vrchol $v$, ze kterého povede alternativní cesta s počátečním časem $n \in \mathbb{N}$. Jestliže vrchol $v$ je v hlavním plánu agenta $i$ dosažitelný po $j$ krocích, pak $n = j+z$, kde $z$ udává dolní mez intervalu výběru (minimální zpoždění). Jeden agent může mít požadavek na více alternativních cest, tudíž je možné, že množina bude obsahovat více prvků.

Řešením upraveného MAPF problému pro každého agenta $i$ je množina obsahující pro každou dvojici $(v, n)$ alternativní plán $\pi_i^a$ z vrcholu $v$ do vrcholu $t(i)$. Přidaná alternativa pro danou dvojici má tvar $\langle j, \langle z, z + k, \pi_i^a\rangle\rangle$.

Upravený MAPF problém vyřešíme znovupoužitím libovolného algoritmu pro řešení klasického MAPF, který drobně pozměníme. Pozměněním rozumíme zahrnutí omezení vyplývající z existence již nalezených cest, rozdílný čas začátku nově přidávaných cest a také skutečnost, že můžeme chtít hledat pro jednoho agenta více cest do cíle z několika startovních bodů. Pro ilustraci se podívejme na úpravu algoritmu CBS pro řešení upraveného MAPF.

Počáteční uzel bude obsahovat omezení týkající se obsazenosti vrcholů v časech určených hlavním řešením plus hlavními bezpečnými intervaly. Doposud jsme

v algoritmu CBS definovali omezení pro agenta $i$, vrchol $v$ a čas $t$. Nyní pro tento účel zavedeme omezení platící opět pro vrchol $v$, čas $t$ a všechny agenty kromě agenta $i$. V rámci své alternativní cesty totiž agent na svou cestu určenou hlavním plánem vstoupit může, aniž by se musel obávat kolize sám se sebou.

Při hledání cesty na nižší úrovni, kde používáme algoritmus A* i při validaci cest na vyšší úrovni v rámci uzlu CBS, pak postupujeme standardně. Tedy kontrolujeme, zda daná cesta není v rozporu s množinou omezení. Navíc pouze ignorujeme konflikt, který by nastal na dvou různých cestách stejného agenta. Spuštěním takto upraveného algoritmu získáme seznam všech alternativ, které dohromady s dříve nalezeným hlavním plánem tvoří plán s alternativami, který jsme chtěli. V případě, že řešení upraveného MAPF neexistuje, pak neexistuje ani striktní alternativně $k$-robustní plán.

### 4.4.4 Provádění plánu

Agent sleduje svou polohu a po příjezdu do vrcholu, ze kterého může odbočit na alternativní cestu, se rozhodne následovně. Spočítá si aktuální zpoždění $d$ a pokud pro něj lze vybrat nějakou z alternativ, tak se touto cestou vydá. Protože v praxi postupujeme tak, že alternativní cesty jsou dostupné k výběru od minimálního zpoždění $d_{min}$ dále, dá se rozhodnutí agenta také vysvětlit následujícím způsobem.

1. $d \leq d_{min}$: Pokračuje dle hlavního plánu, zatím nebyla detekována hrozba $k$-konfliktu.

2. $d_{min} < d \leq k$: Vybírá alternativní cestu, aby $k$-robustně eliminoval konflikt.

3. $k < d \leq k + d_{min}$: Zpoždění agenta přesáhlo mez, kterou jsme na začátku očekávali. V tomto intervalu nicméně stále máme k dispozici výběr validního alternativního plánu (díky podmínce $k$-robustní eliminace). Definice robustnosti chování agenta v tuto chvíli nespecifikuje, v naší verzi implementace tuto alternativu ještě využijeme.

4. $d > k + d_{min}$: Zde už nemáme k dispozici žádnou validní alternativu, agent bude určitě pokračovat dále dle hlavního plánu.

### 4.4.5 Příklad alternativně $k$-robustního plánu

Uvažme graf se dvěma agenty uvedený na obrázku 4.1. Agenti stojí ve svých počátečních vrcholech, cílové vrcholy jsou orámovány příslušnou barvou. Příkladem hlavního plánu může být posloupnost vrcholů, která je na obrázku zobrazena pomocí barevných čísel. Každé číslo udává čas příjezdu agenta označeného stejnou barvou do tohoto vrcholu. V grafu se zobrazeným řešením obsahuje množina konfliktních vrcholů jeden prvek. Na obrázku je vybarven žlutě.

V tomto vrcholu má modrý agent bezpečný interval délky 0 (jeho plánovaný příjezd je v čase 3, příjezd červeného agenta v čase 4). Pro $\forall k \geq 1$ tak budeme konstruovat alternativu $\langle l, \langle min, max \rangle, \pi^a \rangle$. Její začátek bude v našem případě vést z vrcholu, který je označen modrou číslicí 2 – tento vrchol pojmenujeme jako větvící. Zároveň předpokládáme, že modrý agent dosáhne větvícího vrcholu po

dvou krocích, takže nastavíme $l = 2$. Z délky bezpečného intervalu už víme, že $min = 1$.

Konkrétní podoba alternativně $k$-robustní cesty dále záleží na zvoleném $k$. Vzhledem k tomu, že alternativní cesta bude jen jedna, proběhne hledání stejně pro striktní i volnou podobu.

- Pro $k = 1$: hledáme cestu z větvícího vrcholu od času 3 do cíle, která je pro zpoždění o 1 krok 1-robustní. Příklad takové cesty je $\pi^a = 2 - A1 - A2 - 5$. Všimněme si, že při výskytu zpoždění, které zapříčinilo příjezd do větvícího vrcholu v čase 3, se agent do vrcholu označeného $A2$ dostane nejdříve v čase 5, takže podmínka 1-robustnosti platí. Zbývá určit parametr $max$, který v našem případě spočítáme jako $min + k = 1 + 1 = 2$.

  Přidaná alternativa má tvar $\langle 2, \langle 1, 2\rangle, 2 - A1 - A2 - 5\rangle$.

- Pro $k = 2$: opět hledáme cestu z větvícího vrcholu a opět od času 3, protože minimální zpoždění zůstává 1. Cesta musí splňovat podmínky 2-robustnosti a příklad takové cesty je $2 - A1 - A1 - A2 - 5$.

  Vzniklá alternativa je tvaru $\langle 2, \langle 1, 3\rangle, 2 - A1 - A1 - A2 - 5\rangle$.

  Pro $k > 2$ postupujeme analogicky.

### 4.4.6 Vlastnosti

Význam představeného typu robustnosti (a $k$-robustní eliminace) shrnuje následující tvrzení.

**Tvrzení 3.** *Pokud se libovolný z agentů pohybuje při vykonávání hlavního plánu se zpožděním nejvýše k kroků, pak nehrozí kolize s žádným agentem jedoucím po alternativním plánu, jehož celkové zpoždění činí nejvýše $k + 1$ kroků.*

*Důkaz.* Agent, který jede po alternativním plánu, si ho zvolil z důvodu $k$-robustní eliminace $k$-konfliktu. Při výběru muselo činit aktuální zpoždění alespoň jeden krok, jinak by nebylo co eliminovat. Protože alternativní plán je při výběru s minimálním zpožděním $k$-robustní, dovoluje výskyt až $k$ dalších zpoždění, celkem tedy $k + 1$.

Z pohledu agenta na hlavní cestě pak $k$-robustnost vůči minimálnímu zpoždění alternativních cest znamená, že si může dovolit přinejmenším $k$ zpoždění, aniž by se s agentem jedoucím po alternativní cestě mohl srazit. Mezi jeho hlavním plánem a alternativním plánem vybraným s minimálním zpožděním se totiž z definice $k$-robustnosti nenachází $k$-zpožděný konflikt.

□

Volný přístup (bez striktní podmínky) splňuje následující vlastnosti.

**Tvrzení 4.** *Alternativní plány si každý agent může najít nezávisle na ostatních pouze na základě znalosti hlavního řešení.*

*Důkaz.* Plyne z popisu konstrukce plánu a z faktu, že dovolujeme výskyt alternativního konfliktu.

□

*Poznámka.* Tato metoda robustnosti tedy může kombinovat centralizovaný (hledání hlavního řešení) a distribuovaný (hledání alternativních plánů) přístup k řešení. Navíc zde nevzniká nutnost vzájemné komunikace agentů. V případě dostatku výpočetního výkonu tak může teoreticky další alternativní cesty vytvářet za běhu i sám agent.

**Tvrzení 5.** *Pro MAPF problém, ve kterém agenti po dosažení cílového vrcholu mizí z prostředí (disappear at target), existuje ke každému validnímu řešení i alternativně $k$-robustní řešení.*

*Důkaz.* Při eliminaci $k$-konfliktů konstruujeme alternativní plány z vrcholů, které nepatří do množiny konfliktních vrcholů, takže se tu nekříží žádné dva hlavní plány. V rámci alternativní cesty zde agent může čekat libovolně dlouho a zároveň má jistotu, že se v dohledné době uvolní možnost průjezdu původně plánovaným konfliktním vrcholem, protože ostatní agenti si blokují interval s délkou pouze $k$ a to včetně cílového vrcholu (kde po průjezdu zmizí z grafu).

□

*Poznámka.* Pro MAPF, ve kterém agenti zůstávají v cíli *(stay at target)*, toto tvrzení neplatí, viz obrázek 4.2. Červený agent jedoucí z vrcholu 4 do vrcholu 5 má ve vrcholu 3 hlavní bezpečný interval délky 0. Alternativně 1-robustní plán ale neexistuje, protože z vrcholu 4 se nám žádnou alternativu najít nepodaří. To je způsobeno tím, že nemůžeme nijak měnit cestu modrého agenta, který v daném vrcholu již zůstane (je jeho cílový). Tím navždy zablokuje průjezd červenému agentovi.

O plánech, které navíc neobsahují alternativní konflikt (striktní přístup), můžeme říci následující.

**Tvrzení 6.** *Pokud každý z agentů v MAPF problému při provádění plánu s alternativami buď:*

1. *projede každým svým vrcholem v rámci hlavního bezpečného intervalu a nevybere si žádnou alternativní cestu,*

*nebo:*

2. *eliminuje hrozbu prvního nalezeného $k$-konfliktu při minimálním zpoždění a následně projede svým alternativním plánem bez dalšího zpoždění,*

*pak řešení MAPF problému proběhne bez kolizí.*

*Důkaz.* Agenty rozdělíme do dvou skupin – na ty, co splní podmínku *1*, a na ty, co splní podmínku *2*. Toto rozdělení je jednoznačné, protože žádný z agentů nemůže přirozeně splnit obě.

Mezi agenty v první skupině kolize nastat nemůže, což plyne z tvrzení 2. Zároveň víme, že ještě než dojde k přesažení hlavního bezpečného intervalu agentem ve vrcholu, detekujeme nejdříve hrozbu. Pokud ji eliminujeme, pak kvůli ní nemohl nastat konflikt. Že se jedná o první hrozbu je důležité z toho důvodu, že konflikt mohl nastat, pokud bychom našli nějakou hrozbu a z důvodu neexistence větvícího vrcholu mimo množinu konfliktních vrcholů bychom ji neeliminovali.

V případě eliminace první hrozby se ale nic takového nestane a až do přechodu na alternativní plán mezi agenty z libovolných skupin ke konfliktu nedošlo.

Přesuňme se dále do situace, kdy agenti druhé skupiny vykonávají alternativní plán. Mezi nimi ke srážce nedojde, protože jsou zakázány alternativní konflikty. Konečně ke kolizi nemůže dojít ani v případě libovolného agenta z první skupiny a agenta z druhé skupiny, který jede po alternativní cestě. To jednoduše plyne z tvrzení 3, které je dokonce silnější.

□

### 4.4.7 Shrnutí

Představená metoda úpravy plánu si klade za cíl být připravena na dva možné scénáře při jeho provádění. V případě, že se během plánu nevyskytují žádné anomálie, následují agenti svůj optimální plán. Ukáže-li se, že ve skutečnosti dochází k odchylkám od ideální podoby plánu, začne se plán agentů transformovat tak, aby se stal robustním až pro množinu zpoždění velikosti $k$ u každého z nich.

Při konstrukci alternativních plánů dbáme na jejich robustnost vůči hlavním plánům. Tomu také uzpůsobujeme strategii výběru vrcholů, ze kterých vedeme alternativní plány. Požadavek na eliminaci konfliktů pouze ve vrcholech mimo konfliktní množinu nám zaručuje větší pravděpodobnost toho, že se alternativní plán podaří sestrojit a alternativně $k$-robustní řešení bude existovat.

Na základě toho, zda chceme řešit i možné konflikty mezi alternativními plány, rozlišujeme striktní a volný přístup alternativní $k$-robustnosti.

## 4.5 Semi *k*-robustnost

V této sekci se budeme věnovat dalšímu pojetí robustnosti, které konstruujeme pomocí plánů s alternativami. Naše definice bude opět do jisté míry navázána na pojem $k$-robustnosti a také na pojem alternativní $k$-robustnosti. Mírně odlišná definice oproti předchozí metodě nám umožní vytvářet alternativní plány o něco přirozeněji. Začneme s definicí pojmu, který budeme v této sekci používat.

**Definice 23** (bezprostředně předcházející vrchol)**.** *Nechť $\pi_i$ je plán jednoho agenta a nechť $u = \pi_i[k]$ značí libovolný jeho vrchol. Potom vrcholem bezprostředně předcházejícím vrcholu $u$ je vrchol $\pi_i[k-1]$.*

Pro upřesnění dodejme, že v definici uvažujeme vrchol a jeho předchůdce ve smyslu pořadí v plánu. Pokud se tedy v plánu opakovaně vyskytuje tentýž vrchol (ve smyslu polohy v grafu), může mít pokaždé jiný bezprostředně předcházející vrchol. Pro vrchol $\pi_i[0]$ (neboli $s(i)$) bezprostřední předchůdce není definován. Teď už máme vše potřebné pro zavedení nové formy robustnosti.

**Definice 24** (semi $k$-robustnost)**.** *Řešení MAPF problému $\pi$, obsahující plány s alternativami $\pi_1, ..., \pi_n$, nazveme **semi k-robustní**, pokud jsou eliminovány všechny detekované $k$-konflikty a to v bezprostředně předcházejících vrcholech.*

Zde už nepotřebujeme pro agenty hledat alternativní cesty, které jsou $k$-robustní. Stačí nám pouze takové, které jsou pro agenta v danou chvíli bezpečné.

Tímto uvolněním ztratíme garanci další bezpečnosti v případě, že se bude agent dále opožďovat. Na druhou stranu získáme jiné výhodné vlastnosti, které představíme později.

Definicí se snažíme formalizovat požadavek, který se při použití plánování s alternativami pro MAPF zdá být asi nejpřirozenější. Tedy držet se co nejdéle rychlých plánů dle předpokladu a v případě, kdy se vyplní temnější scénáře, odbočit na cestu, která bezprostředně hrozící kolizi odvrátí.

Požadavky na agenty zůstávají stejné jako v případě alternativní $k$-robustnosti.

### 4.5.1 Změny oproti alternativní $k$-robustnosti

Konstrukce semi $k$-robustního plánu bude podobná konstrukci volného alternativně $k$-robustního plánu. Nejdříve zmiňme nejdůležitější odlišnosti.

Ve variantě, kterou popisujeme v předchozí sekci, máme pro vedení alternativního plánu podmínku, která říká, že cesta musí vést z vrcholu mimo množinu konfliktních vrcholů. To je praktické v tom smyslu, že se zde agent může nějaký čas zastavit, aniž by mohl způsobit kolizi.

Na druhou stranu vzniká problém, pokud hlavní řešení $\pi$ obsahuje příliš mnoho prvků z $C_\pi$ na agentově cestě za sebou. Potom rozhodování o výběru alternativy může agent provádět relativně daleko od místa, kde se chce vyhnout kolizi. Čím delší tento úsek máme, tím nám roste pravděpodobnost, že se mezi vrcholem, kde se o změně plánu rozhoduje a vrcholem, kvůli kterému se rozhoduje, objeví další neočekávané zpoždění. Na něj už nelze nijak zareagovat a kolize nastane. Přirozenou volbou, která výše zmíněnou nevýhodu odstraní, je posunout eliminaci konfliktu co nejdále, tedy do bezprostředně předcházejícího vrcholu.

Uvědomme si, proč jsme se v případě alternativní $k$-robustnosti tomuto konceptu vyhnuli. Hledáme-li odbočku z větvícího vrcholu, který je taktéž v množině konfliktních vrcholů, stojíme na „horké půdě“. Nelze zde tudíž dlouho vyčkávat a pokud jsme obklopeni vrcholy rovněž patřícími do množiny konfliktních vrcholů, může se stát, že naše hledání skončí ještě dříve, než začne – nebude možnost odejít ani zůstat.

#### Minimální alternativní interval

Na rozdíl od alternativní $k$-robustnosti nehledáme cesty, které by byly $k$-robustní, nýbrž si vystačíme s cestami, které jsou jen nekonfliktní s cestami ostatních. To ovšem znamená, že konkrétní alternativní plány pro dvě přípustné velikosti zpoždění mohou být různé. Nevytváříme nutně jednu odbočku, ale můžeme jich mít i více na stejném místě hlavního plánu. Volba alternativy tak bude přímo ušitá na míru aktuálnímu zpoždění.

Na druhou stranu může alternativní plán procházet hlavním bezpečným intervalem jiného z agentů. To znamená, že hrozí srážka mezi agentem, který jede bez dalšího zpoždění po své alternativní cestě, a agentem, který se ve svém hlavním plánu zpozdí, byť jen v rámci svého hlavního bezpečného intervalu. Pokud chceme, můžeme tomu zabránit nastavením minimálního alternativního intervalu.

**Definice 25** (minimální alternativní interval (MAI))**.** *Alternativní plány v řešení MAPF zachovávají minimální alternativní interval $min_{alt}$, pokud nemůže nastat*

*kolize agenta jedoucího po hlavním plánu se zpožděním $z \leq min_{alt}$ s agentem jedoucím od okamžiku přechodu na alternativní cestu bez zpoždění.*

V případě, kdy bereme MAI = 0, musíme dávat pozor na výskyt konfliktu vzájemné výměny vrcholů mezi alternativním a hlavním plánem. My při implementaci pracujeme s hodnotu MAI = 1, abychom se tomuto vyhnuli.

Pro alternativní $k$-robustnost jsme představili dvě varianty řešení konfliktů – striktní a volnou. Zde se uchýlíme k myšlence volného přístupu, který alternativní konflikty neřeší. V případě striktního přístupu bychom narazili na problémy s neexistencí alternativních cest, které plynou ze strategie výběru větvících vrcholů. Navíc v případě semi $k$-robustnosti budeme spíše předpokládat, že výskyt zpoždění je okrajová záležitost a alternativní plán si vybere jen minimum agentů.

### 4.5.2 Konstrukce plánu

Stejně jako v případě konstrukce alternativně $k$-robustního plánu začneme s nastavením hlavních bezpečných intervalů a sestrojíme množinu $B$ obsahující vrcholy s hlavním bezpečným intervalem délky menší než $k$.

Pro přidání alternativních cest do plánu opět použijeme prohledávací algoritmus A*. Jednotlivé prvky bereme z množiny $B$ postupně jeden po druhém. Popíšeme přidání alternativního plánu na základě libovolného z nich. Nechť se týká agenta $i$ a vrcholu $v$ s délkou bezpečného intervalu $d$ (platí, že $d < k$).

Nejprve najdeme vrchol $u$, ze kterého povede alternativní cesta. Pokud takový (dle definice 23) neexistuje, pak tento alternativní plán konstruovat nebudeme a pokračujeme na dalším prvek z $B$. Jinak začneme hledat cestu z $u$ do cílového vrcholu agenta $i$. Pro každou velikost zpoždění $z = d+1,..., k$ ji hledáme separátně od času $t_u + z$, kde $t_u$ značí plánovaný příjezd agenta do vrcholu $u$. Při generování následníků v A* bereme v potaz časy plánovaných příjezdů ostatních agentů v hlavním řešení a omezení dané velikostí MAI. Interval výběru každé nalezené alternativní cesty bude tvaru $\langle z, z \rangle$.

Vypořádáme-li se takto se všemi prvky $B$, jsme hotovi. Pokud v nějakém bodě zjistíme, že cesta z vrcholu $u$ neexistuje, potom skončíme neúspěchem.

### 4.5.3 Provádění plánu

Agent pohybující se prostředím se v každém vrcholu rozhoduje podle následujícího schématu.

1. Zjistí svoje aktuální zpoždění.

2. Je-li pro toto zpoždění k dispozici alternativa, změní svůj plán takovým způsobem, aby v dalším kroku pokračoval do již zmíněné alternativní větve. Jinak pokračuje dle hlavního plánu.

### 4.5.4 Příklad semi *k*-robustního plánu

Nyní si názorně ukážeme, jak vypadá semi $k$-robustní plán v grafu se dvěma agenty, který je vyobrazen na obrázku 4.1. Jak je vidět, množinu konfliktních vrcholů řešení MAPF tvoří žlutě vybarvený vrchol a problém má modrý agent,

neboť jeho hlavní bezpečný interval v tomto vrcholu má délku 0. Jako větvící vrchol vybereme vrchol bezprostředně předcházející, což je vrchol označený modrou číslicí 2. Dále už konstrukce alternativních cest závisí na parametru $k$.

- Pro $k = 1$: hledáme nekonfliktní alternativní cestu z větvícího vrcholu v čase 3 do cíle. Příkladem takové cesty je cesta $2 - A1 - A2 - 5$.

  Přidaná alternativa má podobu $\langle 2, \langle 1, 1\rangle, 2 - A1 - A2 - 5\rangle$.

- Pro $k = 2$: hledáme nekonfliktní alternativní cestu z větvícího vrcholu v čase 3 a v čase 4. Bude-li při provádění plánu zpoždění délky 1, můžeme použít cestu z předchozího bodu $(2 - A1 - A2 - 5)$.

  Při zpoždění délky 2 už můžeme jako alternativní použít původní cestu $2 - 3 - 4 - 5$, protože čas příjezdu do vrcholu s modrou číslicí 3 bude až v čase 5. Pozor, to platí jen v případě, kdy je MAI = 0. Počítáme-li s hodnotou MAI = 1, pak alternativní cestu $2 - 3 - 4 - 5$ vzít nemůžeme a musíme vzít např. posloupnost $2 - 2 - 3 - 4 - 5$.

  Do plánu přidáváme dvě alternativy $\langle 2, \langle 1, 1\rangle, 2 - A1 - A2 - 5\rangle$ a $\langle 2, \langle 2, 2\rangle, 2 - 3 - 4 - 5\rangle$ (pro MAI = 0).

  Pro $k > 2$ postupujeme analogicky.

### 4.5.5 Vlastnosti

**Tvrzení 7.** *Nechť se libovolný agent $i$ zpozdí během exekuce plánu o nejvýše $k$ kroků a ostatní agenti jedou na čas (bez zpoždění). Potom každý konflikt, který mohl nastat vlivem zpoždění v klasickém validním řešení MAPF mimo vrchol $s(i)$, bude odstraněn.*

*Důkaz.* Každý vrchol v plánu agenta $i$ (kromě $s(i)$) má svého předchůdce. Z tvrzení 2 vyplývá, že ke kolizi může za podmínek aktuálně dokazovaného tvrzení dojít pouze přesažením hlavního bezpečného intervalu některého z vrcholů v plánu $\pi_i^h$. Ještě než k tomu dojde, bude tato skutečnost detekována jako hrozba $k$-konfliktu. Protože semi $k$-robustní plán všechny hrozby eliminuje, dojde k výběru alternativní cesty, která kolizi zamezí.

□

*Poznámka.* Pokud navíc agent projede alternativní plán bez dalšího zpoždění, bude exekuce bezkolizní. V opačném případě může samozřejmě dojít ke kolizi tohoto agenta s jiným agentem jedoucím po hlavním plánu.

Dále dokažme ještě jedno tvrzení týkající se délky plánu. Toto tvrzení platí v případě, kdy máme MAI rovný nule.

**Tvrzení 8.** *Nechť se libovolný agent $i$ zpozdí během exekuce plánu o nejvýše $k$ kroků a ostatní agenti jedou na čas. Pak v každém okamžiku exekuce plánu jede agent $i$ do cíle cestou, která je v daném okamžiku bezkolizní a nejkratší možná vůči původnímu plánu.*

*Důkaz.* Následuje-li agent $i$ po celou dobu svůj hlavní plán, jedná se o plán zachovávající nejlepší hodnotu účelové funkce, protože jsme k jeho nalezení použili

řešič, který dává optimální řešení a je úplný. V tomto smyslu ho bereme jako nejkratší možný.

Zvolil-li agent alternativní cestu, pak na základě předchozích tvrzení detekoval hrozbu $k$-konfliktu, která měla nastat v příštím kroku (a protože se ostatní agenti nezpozdí, tak by nastala). Nová cesta byla konstruována jako nejkratší možná vzhledem k danému zpoždění a zabránění kolizi. K nalezení byl použit algoritmus vracející nejkratší cestu.

□

### 4.5.6 Shrnutí

V případě semi $k$-robustnosti se opět snažíme pomocí konstrukce alternativních cest vypořádat se zpožděními agentů až do velikosti $k$. Vlastnosti sledujeme spíše z pohledu jednotlivých agentů. Alternativní plány konstruujeme vždy pro pevně danou délku zpoždění.

Jedná se o techniku, která řeší možné konflikty těsně předtím, než nastanou. Na rozdíl od alternativní $k$-robustnosti dokáže pro pevně stanovené číslo $k$ eliminovat všechny hrozby $k$-konfliktů, které se objeví. Vzhledem ke strategii výběru vrcholů pro alternativní cesty je více náchylná k tomu, že řešení nebude existovat.

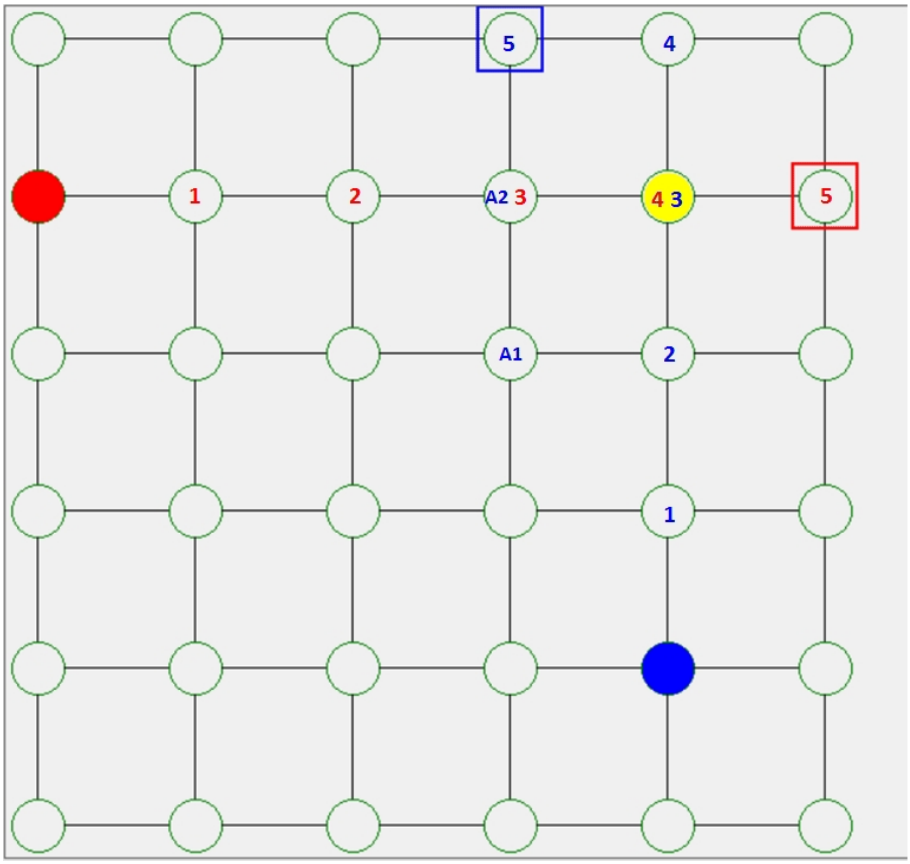
## 4.6 Srovnání

Na závěr kapitoly ještě pomocí tabulky 4.1 srovnejme, jakými nástroji se snaží představené přístupy zabránit vzniku konfliktů. Konflikty rozlišíme dle místa vzniku na tři druhy – *hlavní+hlavní* (oba agenti se srazí na cestě v rámci hlavního plánu), *hlavní+vedlejší* (do agenta provádějícího hlavní plán narazí agent jedoucí po alternativní cestě nebo naopak) a *vedlejší+vedlejší* (srážka agentů proběhla na jejich alternativních cestách).

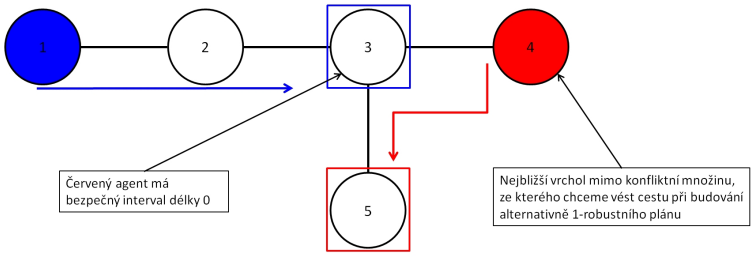
| **Způsob ochrany před jednotlivými druhy konfliktů** | | | |
|---|---|---|---|
| | *hlavní+hlavní* | *hlavní+vedlejší* | *vedlejší+vedlejší* |
| **Striktní alternativní $k$-robustnost** | cesty jsou validní, možnost přesunu na alternativní cesty | cesty jsou $k$-robustní [1] | cesty jsou validní [1] |
| **Volná alternativní $k$-robustnost** | cesty jsou validní, možnost přesunu na alternativní cesty | cesty jsou $k$-robustní [1] | neřeší |
| **Semi $k$-robustnost** | cesty jsou validní, možnost přesunu na alternativní cesty | cesty jsou validní, možnost nastavení MAI | neřeší |

Tabulka 4.1: Srovnání technik dle způsobu ochrany před jednotlivými druhy konfliktů.

[1] Při výběru s minimálním zpožděním.




Obrázek 4.1: Příklad konstrukce alternativně a semi $k$-robustních plánů




Obrázek 4.2: Příklad neexistence alternativně 1-robustního plánu.

# 5. Zvýšení robustnosti pomocí přidání *wait* akcí

V předchozí kapitole jsme pro zvýšení robustnosti přidávali do plánu alternativní cesty, které měly za úkol odklánět agenty do potenciálně rizikových vrcholů. Nyní se budeme snažit o vytvoření robustního plánu, který nezvýší hodnotu účelové funkce ani nijak nezmění vrcholy, kterými agenti prochází. Dosáhneme toho pouze přidáváním *wait* akcí do nalezeného plánu.

## 5.1 Popis metody

Při posuzování kvality plánů používáme často účelovou funkci makespan. Tato hodnota zpravidla závisí na jednom, případně několika málo agentech, jejichž délka cesty je maximální. Ostatní agenti mají cestu kratší a po příjezdu do cílového vrcholu čekají na dokončení plánů ostatních. Dále se budeme zabývat myšlenkou přesunu *wait* akcí z cílového vrcholu do vrcholů po cestě a jeho vlivu na celkovou robustnost řešení.

Nechť $\pi$ je validní plán pro $n$ agentů a $\pi_i$ označme plán $i$-tého z nich. Na začátek zavedeme několik pojmů.

**Definice 26** (minimální průjezd vrcholem)**.** *Pro agenta $i$ a jeho libovolný vrchol $v_k = \pi_i[k]$, dosažený po $k$ krocích, je minimální průjezd vrcholem $v_k$ počet move akcí z celkového počtu $k$ kroků. Značíme $min_{\pi_i}(v_k)$.*

**Definice 27** (maximální průjezd vrcholem)**.** *Pro agenta $i$ a jeho libovolný vrchol $v_k = \pi_i[k]$, dosažený po $k$ krocích je maximální průjezd vrcholem $v_k$ definován jako: $max_{\pi_i}(v_k) = makespan_\pi - min_{\pi_i}(t(i)) + min_{\pi_i}(v_k)$.*

**Definice 28** (plánovaný průjezd vrcholem)**.** *Pro agenta $i$ a jeho libovolný vrchol $v_k = \pi_i[k]$, dosažený po $k$ krocích je plánovaný průjezd vrcholem $v_k$ roven číslu $k$.*

Minimální a maximální průjezd vrcholem definuje dolní a horní mez časového intervalu, ve kterém se agent musí ve vrcholu objevit, pokud má dodržet posloupnost vrcholů v plánu i původní hodnotu účelové funkce. Kromě vrcholu rozlišujeme i pořadí, takže pro stejný vrchol lze dostat různé hodnoty, pokud se tento vrchol v plánu opakuje a to nikoliv bezprostředně za sebou. Označením $t(i)$ rozumíme vrchol, ze kterého už nepokračujeme žádnou *move* akcí. Toto upřesnění uvádíme vzhledem k tomu, že pracujeme s konceptem *stay at target*, takže některý agent může v rámci svého plánu přejít do cílového vrcholu a v pozdějším čase ho opustit kvůli průjezdu jiného agenta, aby se do něj následně vrátil.

Pro ilustraci uvádíme v tabulce 5.1 konkrétní hodnoty pro cestu agenta $i$ v nějakém MAPF řešení $\pi$. Předpokládáme, že agent má dle svého plánu absolvovat cestu v části grafu uvedené na obrázku 5.1 z vrcholu $S$ do $T$, přičemž daná posloupnost navštívených vrcholů má tvar $S-A-A-B-C-B-D-T-T-T-T$. Hodnota funkce makespan řešení $\pi$ je v tomto případě rovna číslu 10. Při použití řešiče založeného na hledání nejkratších cest (A*, CBS) bude zpravidla plánovaný čas průjezdu blízko minimálnímu průjezdu.

Nyní se pojďme věnovat místům, ve kterých hrozí případné kolize. Jak jsme již popsali dříve, jedná se o vrcholy, kterými projedou alespoň dva agenti. Zároveň při zvolení minimální délky bezpečného intervalu $b$ (dle definice 17), který bychom chtěli v řešení mít, dostaneme množinu *porušení bezpečných intervalů*. Porušení znamená, že agent $i$ nemá ve vrcholu $v$ dostatečnou rezervu $b$ kroků, protože se zde očekává příjezd dalšího agenta $j$. Vzniklý problém bychom mohli odstranit, pokud by šlo posunout plán agenta $j$ pomocí *wait* akcí tak, aby do inkriminovaného vrcholu $v$ přijel o něco déle. Tím se agentu $i$ natáhne délka bezpečného intervalu a riziko kolize se výrazně sníží. Abychom věděli, kolik časových úseků je možné v libovolném vrcholu čekat, zavedeme pojem maximální čekací doby.

**Definice 29** (maximální čekací doba)**.** *Pro agenta $i$ a libovolný vrchol $v_k = \pi_i[k]$, dosažený po $k$ krocích je maximální čekací doba agenta ve vrcholu rovna minimu z čísel $b(v_k) - b$, $max_{\pi_i}(v_k) - k$, kde $b(v_k)$ je délka bezpečného intervalu agenta $i$ ve vrcholu $v_k$.*

Maximální čekací doba nám označuje nejvyšší možný počet *wait* akcí, které můžeme agentovi ve vrcholu přidat nad rámec původního plánu. Chceme přitom dodržet minimální bezpečný interval a zároveň nenavyšovat původní makespan řešení.

Při zvýšení robustnosti se soustředíme na zvýšení hodnoty bezpečného intervalu pro všechny agenty ve všech vrcholech na hodnotu alespoň $b$. Pokud se toto navýšení podaří, máme $b$-robustní plán. Zpravidla se ale vyřešení všech problémů nedá očekávat, takže tuto metodu je vhodné použít spíše jako doplňkovou. Nám se hodí pro zvýšení robustnosti hlavního řešení před konstrukcí alternativních cest v případě plánů s alternativami.

## 5.2 Předpoklady

Metoda na vstupu předpokládá jakékoliv validní řešení MAPF problému $\pi$. Při práci s bezpečným intervalem agenta ve vrcholu pak nastavujeme horní hranici intervalu u agenta, který projede vrcholem jako poslední na hodnotu nekonečno.

## 5.3 Konstrukce plánu

Nechť máme zadaný validní plán $\pi$, ze kterého při konstrukci (nebo spíše lokální úpravě) vycházíme. Pro každého agenta a vrchol v plánu spočteme hodnoty minimálního a maximálního průjezdu a určíme rovněž maximální čekací dobu.

Dále máme zadanou hodnotu minimálního bezpečného intervalu $b$, kterou bychom chtěli v plánu mít. Nalezneme všechny čtveřice $\langle i, j, t, d\rangle$ popisující situaci, kdy má agent $i$ v čase $t$ bezpečný interval pouze $d$, a to z důvodu příjezdu agenta $j$ v čase $t + d + 1$.

Čtveřice seřadíme vzestupně dle hodnoty $d$ a začneme je postupně zpracovávat. Pro libovolnou čtveřici $\langle i, j, t, d\rangle$ posuneme příjezd agenta $j$ do vrcholu $\pi_j[t+d+1]$ až o $b-d$ kroků. Konkrétní hodnotu tohoto posunu (označíme ji jako $h$) určíme na základě následujících podmínek.

1. Číslo $h$ musí být menší nebo rovno hodnotě maximální čekací doby z vrcholu $\pi_j[t+d]$. To nám zaručí, že přidání *wait* akcí do tohoto vrcholu je možné a také není dotčen průběh plánu u žádného z předchůdců vrcholu $\pi_j[t+d]$.

2. Musí platit, že $h$ je menší nebo rovno hodnotě maximální čekací doby všech vrcholů $\pi_j[x]$ pro $x \geq t+d+1$ až do chvíle, kdy je těmito vrcholy absorbováno $h$ čekacích akcí.

   Vrchol $v$ *absorbuje* $w$ *wait* akcí, pokud v něm agent v původním plánu čeká právě $w$ kroků. Čekáním rozumíme provedení *wait* akce. Cílový vrchol $t(i)$ agenta $i$ na počátku absorbuje tolik *wait* akcí, kolik udává hodnota jeho maximální čekací doby.

   Pokud některý z vrcholů absorbuje jen $h' < h$ *wait* akcí, pak se požadavek na všechny jeho následníky snižuje na hodnotu $h - h'$.

Pro každou čtveřici chceme prodloužit příjezd agenta $j$ o nejvyšší možnou hodnotu $h$ splňující požadavky výše. Číslo $h$ a vrchol, od kterého se bude přidávat, si pro každou čtveřici poznamenáme. Po zpracování všech čtveřic zkonstruujeme na základě poznámek o počtu přidaných *wait* akcí nový robustnější plán.

Konstrukci takového plánu provedeme postupně pro každého agenta zvlášť. Nejdříve se podíváme na požadavky o navýšení počtu *wait* akcí v jednotlivých vrcholech. Jestliže se na nějaký vrchol vztahuje $p$ různých požadavků, výsledný počet přidaných *wait* akcí bude odpovídat nejvyššímu z nich.

Není těžké nahlédnout, že takový popis konstrukce je korektní. Vzhledem k tomu, že *wait* akce přidáváme vždy od určitého vrcholu až do cíle (takže pouze jedním směrem) a pro každý jednotlivý požadavek vztahující se k určité čtveřici byla úprava korektní, musí být korektní i pro ten největší z nich. Navíc pokud jsme chtěli vyřešit nějaký problém posunem příjezdu do vrcholu $v$ o $k$ kroků a jiný problém posunem o $l$ kroků (kde $k < l$), pak celkovým posunem o $l$ kroků vyřešíme problémy oba.

## 5.4 Vlastnosti

Nejdůležitější vlastnost se týká délky upraveného plánu.

**Tvrzení 9.** *Hodnota účelové funkce makespan původního a upraveného plánu je stejná.*

*Důkaz.* Plyne z popisu konstrukce nového plánu.

□

Další tvrzení říká, že odstraněním jednoho porušení bezpečného intervalu nevytvoříme jiné.

**Tvrzení 10.** *Žádná úprava plánu robustnosti pomocí přidání wait akcí nevytvoří nový potenciální konflikt, který by mohl nastat při zpoždění některého agenta o méně než b kroků (při zvolené minimální hodnotě bezpečného intervalu b).*

*Důkaz.* Místa, ve kterých může dojít ke kolizi vlivem zpoždění některého z agentů o méně než $b$ kroků, jsme při konstrukci robustního plánu našli a snažili se je

odstranit. V rámci tohoto procesu došlo buď ke zvýšení hodnoty bezpečného intervalu, nebo bylo zachována původní hodnota.

Pokud jsme do plánu některého z agentů přidali *wait* akce, posunovali jsme se v rámci bezpečného intervalu tohoto agenta v daném vrcholu a vždy jsme se ujistili, že jeho délka zůstala alespoň $b$.

□

Nakonec ukážeme, že úpravu plánu popsanou v konstrukci umíme provést v rozumném čase.

**Tvrzení 11.** *Složitost algoritmu je* $\mathcal{O}(n^2 \cdot makespan_\pi)$.

*Důkaz.* Výpočet pomocných hodnot (minimální, maximální průjezd vrcholem, maximální čekací doba) zvládneme v čase $\mathcal{O}(n \cdot makespan_\pi)$, kde $n$ je počet agentů. Počet čtveřic porušující interval $b$ závisí na hodnotě $b$ a velikosti množiny konfliktních vrcholů. Zpravidla by neměl být velký, nicméně v nejhorším případě může být čtveřic až $\mathcal{O}(n^2 \cdot makespan_\pi)$. Zpracování jedné takové čtveřice zvládneme v konstantním čase, pokud si předem spočítáme příslušné hodnoty (složitost této operace je $\mathcal{O}(n \cdot makespan_\pi)$). Stejnou dobu zabere konstrukce nového plánu.

Celková složitost tak vychází $\mathcal{O}(n^2 \cdot makespan_\pi)$.

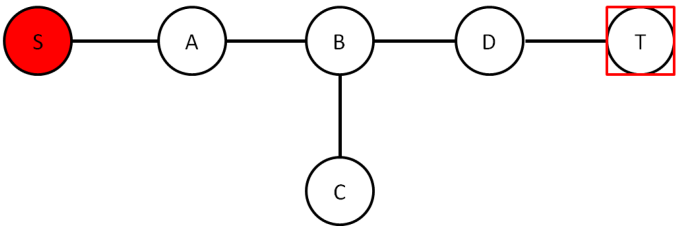
□

## 5.5 Shrnutí

Jedná se o doplňkovou a časově nepříliš náročnou techniku, která zvýší odolnost optimálního plánu vůči případným kolizím, které mohou být způsobeny zpožděním agentů při vykonávání plánu. Nijak nemění celkovou délku plánu, ani posloupnost navštívených vrcholů, takže v tomto ohledu je jejím výstupem plán, který je ekvivalentní klasickému validnímu plánu. Proto tuto metodu při testování využijeme v kombinaci s alternativní $k$-robustností a semi $k$-robustností. Upravený plán s přidanými *wait* akcemi zde použijeme jako hlavní plán.

| $v$ | čas plánovaného průjezdu | $min_{\pi_i}(v)$ | $max_{\pi_i}(v)$ |
|---|---|---|---|
| $S_0$ | 0 | 0 | 3 |
| $A_1$ | 1 | 1 | 4 |
| $A_2$ | 2 | 1 | 5 |
| $B_3$ | 3 | 2 | 6 |
| $C_4$ | 4 | 3 | 7 |
| $B_5$ | 5 | 4 | 8 |
| $D_6$ | 6 | 5 | 9 |
| $T_7$ | 7 | 6 | 10 |
| $T_8$ | 8 | 6 | 10 |
| $T_9$ | 9 | 6 | 10 |
| $T_{10}$ | 10 | 6 | 10 |

Tabulka 5.1: Hodnoty minimálního a maximálního průjezdu pro vrcholy plánu jednoho agenta.




Obrázek 5.1: Graf pro cestu červeného agenta.

# 6. Robustnost pomocí změny rychlosti agentů

Robustnost plánů se nám hodí v případě, kdy tušíme, že jeho vykonávání nemusí probíhat tak hladce, jak jsme teoreticky spočítali. V situacích, kdy se potýkáme se zpožděním agenta oproti plánu, si je dobré uvědomit, že při fyzickém provedení k němu může docházet odlišněji.

Ve chvíli, kdy simulujeme řešení klasického MAPF problému, ve kterém se zpožděním nepočítáme, dává dobrý smysl zaměřit se na jednoduchost provedení. Cesty agentů prostředím tedy bereme jako posloupnost vrcholů grafu, mezi kterými agenti přeskakují. Zpoždění v tomto modelu pak znamená „zamrznutí“ v aktuálním vrcholu.

Zajímá-li nás ale především informace o pravděpodobnosti možných kolizí, může se tato abstrakce jevit jako zavádějící. Agenti se ve skutečnosti pohybují mnohem plynuleji, takže tráví většinu času na hranách mezi vrcholy. Agenti se srazí, přiblíží-li se navzájem na vzdálenost, která je menší než součet poloměrů jejich velikostí, což může být výrazně méně než uvažovaná vzdálenost mezi dvěma vrcholy grafu.

My nyní představíme realističtější pojetí, ve kterém počítáme i s dobou, kterou agent tráví na cestě mezi danými vrcholy. Zjemníme časové úseky, což nám umožní sledovat chování agenta během cesty po hraně mezi vrcholy. Zároveň nám to dovolí plynule snižovat a zvyšovat rychlosti všech agentů.

## 6.1 Popis metody

Pro zjednodušení uvažujme, že všechny hrany mají stejnou váhu. Zvýšení robustnosti provedeme pomocí zjemňování časových úseků. Předpokládejme, že cesta z jednoho vrcholu do druhého trvá nějakou dobu $t$. V klasickém MAPF problému odpovídá tato doba jednomu časovému úseku. Po $k$-násobném zjemnění bude cesta z jednoho vrcholu do druhého (doba $t$) odpovídat $k$ časovým úsekům (krokům). Jestliže měl agent původně projet vrcholem $A$ v čase 1 a vrcholem $B$ v čase 2, pak po zjemnění dostaneme průjezd vrcholem $A$ po $k$ krocích a průjezd vrcholem $B$ po $2k$ krocích. Časové úseky $(k+1,...,2k-1)$ stráví agent přejezdem mezi $A$ a $B$.

Dále si musíme uvědomit, že kolize při provádění plánu vznikají zpožďováním agentů oproti předpokladu plánovače. Situace se postupem času čím dál tím více zhoršuje, protože agenti už nemají šanci své zpoždění dohnat. Proto povolíme možnost zrychlení agentů, aby měli příležitost své zpoždění korigovat a napojit se opět na původní plán. Zároveň, aby bylo možné zrychlovat, je nutné nejdříve zpomalit. U agentů proto nastavíme počáteční minimální rychlost, se kterou začnou své plány vykonávat. Jako maximální rychlost pak bereme tu, se kterou by agenti vykonávali plány v běžném případě – tedy projetí cesty maximální rychlostí trvá stejný čas jako vykonání optimálního plánu v klasickém MAPF.

Klíčovým prvkem, který sledujeme, je velikost procentuálního nárůstu času při projetí hrany minimální rychlostí namísto maximální. A právě na základě toho definujeme pojem $min/max$ robustnost.

**Definice 30** ($min/max$ robustnost)**.** *Plán $\pi = \{\pi_1,...,\pi_k\}$ nazveme* ***min/max robustní****, pokud je validní a zároveň je rychlost každéhé agenta nastavena tak, aby cesta z vrcholu do jeho souseda trvala max jednotek času. Navíc ale musí být agent schopen zvýšit svoji rychlost tak, aby mohl daný úsek projet v čase min.*

K upřesnění pojmu rychlost přidáme ještě formální definici.

**Definice 31** (rychlost agenta v $min/max$ robustním plánu)**.** *Agent se v min/max robustním plánu pohybuje (aktuální) rychlostí $r$, pokud mu průjezd hranou grafu trvá právě $\frac{max}{r}$ časových úseků, přičemž požadujeme, aby $\frac{max}{r} \in \mathbb{N}$.*

V každém $min/max$ robustním plánu tedy máme na výběr rychlosti z množiny $\mathcal{R} = \{r : 1 \leq r \leq \frac{max}{min} \wedge \frac{max}{r} \in \mathbb{N}\}$.

Požadovanou míru robustnosti lze zajistit volbou čísel $min$ a $max$. Čím větší je podíl příslušných hodnot, tím větší dokážeme umožnit zrychlení agenta oproti plánu v případě, kdy začne nabírat zpoždění. V optimálním scénáři agent téměř okamžitě dožene ztrátu způsobenou předchozím zpožděním a do vrcholů dojede v souladu se svým plánem, čímž nedojde k žádným kolizím.

## 6.2 Předpoklady

Na začátku vycházíme z libovolného validního řešení klasického MAPF problému. Pro zavedení robustnosti potřebujeme informace o aktuální poloze agentů a míře zpoždění ve chvíli, kdy se nachází v nějakém vrcholu. Dále potřebujeme agenty, kteří disponují schopností měnit svou rychlost.

Samotná realizace může být provedena jak čistě centrálním, tak i distribuovaným způsobem. V prvním případě musí hlídat polohu i zpoždění plánovač. Agenti pak pouze přijímají zprávy, podle kterých mění svoji rychlost. V distribuovaném pojetí si informace o poloze a času získává sám agent prostřednictvím svých senzorů a lokálně probíhá rovněž výpočet aktuální rychlosti.

## 6.3 Konstrukce plánu

Na vstupu vezmeme validní plán ze základní definice MAPF problému. Ten nám poskytne posloupnost vrcholů spolu s předpokládanými časy jejich navštívení. My z uvedených plánů zjemněním a úpravou rychlosti sestavíme každému agentovi $min/max$ robustní plán, který bude obsahovat časy příjezdů do vrcholů po provedení zjemnění. Pro ilustraci uvádíme v tabulce 6.1 příklad rozvržení času takového plánu pro jednoho agenta a jeho podobu po trojnásobném zjemnění. Sestavením zjemněných plánů pro $k$ agentů dostaneme plán $\pi_{min/max}$.

| | S | A | B | T |
|---|---|---|---|---|
| Původní plán | 0 | 1 | 2 | 3 |
| Plán po 3-násobném zjemnění | 0 | 3 | 6 | 9 |

Tabulka 6.1: Časy příjezdů do jednotlivých vrcholů pro plán s posloupností vrcholů $S - A - B - T$.

## 6.4 Provádění plánu

Nechť máme sestavený zjemněný plán $\pi_{min/max}$. Každý z agentů má dále určenou množinu povolených rychlostí, kterými se může v grafu pohybovat.

Svůj plán začne agent vykonávat s minimální povolenou rychlostí. Při projetí každého z vrcholů zjistí, kolik činí zpoždění vůči předpokladu a zvolí novou vhodnou rychlost k projetí další hrany tak, aby se toto zpoždění co nejvíce snížilo, ale agent zároveň nedorazil do dalšího vrcholu dříve, než by měl dle zjemněného plánu.

V tabulce 6.2 uvádíme konkrétní příklad exekuce 2/3 robustního plánu jednoho z agentů. Zápis jeho polohy ve zjemněném plánu uvádíme ve tvaru $v + x$, kde $x \in \langle 0{,}1)$. To znamená, že daný agent jede z vrcholu $v$ do svého následníka $n$ a na hraně z $v$ do $n$ urazil podíl $x$. Takže zápis $v + 0{,}25$ popisuje, že se agent nachází ve čtvrtině délky hrany spojující vrchol $v$ s vrcholem, do kterého agent jede.

V příkladu uváděném v tabulce 6.2 nastalo zpoždění v čase 4, což vedlo ke zvýšení rychlosti po příjezdu do vrcholu $B$ na hodnotu 1,5. Pomocí tohoto zrychlení dokázal agent zpoždění smazat a dosáhnout cíle v předem plánovaný čas.

| Čas | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| Poloha | S+0 | S+$\frac{1}{3}$ | S+$\frac{2}{3}$ | A+0 | A+$\frac{1}{3}$ | A+$\frac{1}{3}$ | A+$\frac{2}{3}$ | B+0 | B+$\frac{1}{2}$ | T |
| Zpoždění | ne | ne | ne | ne | ano | ne | ne | ne | ne | ne |
| Rychlost | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1,5 | 1,5 | – |

Tabulka 6.2: Příklad exekuce 2/3 robustního plánu $S - A - B - T$.

## 6.5 Detekce konfliktů

Protože se agenti během provádění plánu mohou v jednotlivých časových úsecích nacházet i na hranách mezi vrcholy, uvedeme specifikaci toho, co v tomto případě považujeme za vrcholový konflikt a konflikt výměny vrcholů.

- **Vrcholový konflikt**: dva agenti s polohami $v_1{+}x_1$ a $v_2{+}x_2$ mají v $min/max$ robustním plánu vrcholový konflikt, pokud platí, že $v_1 = v_2$, a zároveň $|x_1 - x_2| < \frac{1}{max}$.
- **Konflikt vzájemné výměny vrcholů**: dva agenti s polohami $v_1 + x_1$ a $v_2 + x_2$ jedoucí do vrcholů $w_1$ a $w_2$, mají v $min/max$ robustním plánu konflikt výměny vrcholů, pokud platí, že $v_1 = w_2$ a zároveň $v_2 = w_1$, bez ohledu na hodnoty $x_1$ a $x_2$. Pokud totiž míří po stejné hraně opačným směrem, musí mezi nimi dojít ke srážce.

## 6.6 Vlastnosti

Podívejme se, jaká tvrzení platí o $min/max$ robustních plánech.

**Tvrzení 12.** *Délka min/max robustního plánu bude bez započítání vlivu případného zpoždění nejvýše* $\frac{max}{min}$ *krát delší než délka optimálního plánu.*

*Důkaz.* Nedojde-li během exekuce ke zpoždění, bude projetí jedné hrany trvat čas $max$. V případě klasického plánu by agenti projeli tuto hranu za čas $min$.

Vliv zpoždění na celkovou délku plánu je stejný jako u klasického validního plánu. V případě $min/max$ robustnosti se navíc předpokládá jeho odstranění po cestě (pokud se nejedná o přejezd do cílového vrcholu), takže v poměru ke klasickému provedení plánu bude reálně ještě o něco menší.

□

**Tvrzení 13.** *Nezpozdí-li se žádný agent na přejezdu mezi dvěma sousedními vrcholy o více než* $(max - min)$ *časových úseků, nenastane v plánu kolize.*

*Důkaz.* Každý agent $i$ má rozvrženou cestu obsahující $v_i$ vrcholů. $J$-tým vrcholem pak dle rozvrhu projede v čase $j \cdot max$. Svoji rychlost mění dle hodnoty zpoždění v každém vrcholu, přičemž začíná s nulovým zpožděním. Pokud dojde při cestě mezi vrcholy, které si agent naplánoval urazit za $k$ časových úseků ke zpoždění o $z$ úseků, odpovídá to vlastně jízdě bez zpoždění s časem $k + z$.

Agent vždy plánuje rychlost tak, aby do dalšího vrcholu dorazil bez zpoždění. Z něj se následně chystá pokračovat minimální rychlostí s délkou jízdy $max$ úseků. Dojde-li k neočekávanému zpoždění délky $d \leq max - min$, bude po příjezdu do vrcholu činit právě $d$. A pro každou hodnotu $d \leq max - min$ lze zvolit rychlost tak, aby plánovaný příjezd do dalšího z vrcholů byl opět načas.

Z toho plyne, že v takovém případě se žádný agent nikdy nedostane do situace, kdy by jeho zpoždění vůči rozvrhu činilo více než $max - min$ času.

Protože původní plán nemá vrcholový konflikt, jsou mezi agenty plánované rozestupy alespoň $max$. Dále platí, že žádný agent se oproti rozvrhu nepředchází. Z uvedeného vyplývá, že při exekuci zůstane mezi agenty zachován odstup nejméně $min$ časových úseků a ke srážce nemůže dojít.

□

## 6.7 Shrnutí

Představili jsme nový koncept založený na odlišném vnímání polohy agenta v čase. Snažíme se v něm pracovat s dobou, kterou agenti reálně stráví na hranách grafu během projíždění mezi jednotlivými vrcholy jejich plánu.

Ukázali jsme, že plány konstruované technikou *min/max* robustnosti jsou robustní vůči pravidelně se opakujícímu výskytu zpoždění. Nepřesáhne-li během daného časového úseku určitou mez, dokážeme se v plánech ze zpoždění zotavit. Míra zvýšení robustnosti se pak přímo projeví ve zvětšení délky plánu.

# 7. Experimenty

V předchozích kapitolách jsme se věnovali různým způsobům, které měly zajistit robustnost řešení v problémech MAPF. Nabízí se jistě otázka, do jaké míry jsou představené metody použitelné a vhodné.

V této kapitole se budeme zabývat testováním kvality všech způsobů robustnosti, které byly v práci představeny. Postupně se podíváme na výsledky navržených technik a přidáme i referenční hodnoty pro klasické validní řešení a řešení pomocí již dříve známé $k$-robustnosti. Nakonec provedeme i grafické srovnání všech zmíněných robustních technik mezi sebou.

Při testech musíme zohlednit vliv mnoha parametrů, podle kterých lze kvalitu hodnotit. Představme si nyní parametry, které mají na provádění MAPF benchmarků největší vliv.

### Výběr prostředí

Nutnou podmínkou pro testování je výběr vhodného grafu, ve kterém se agenti budou pohybovat. Jak jsme již zmínili v úvodní kapitole práce, lze všechny použité techniky aplikovat v libovolném grafu $G$, kde se jednotlivé hrany kříží pouze ve vrcholech a mají stejnou vzdálenost.

V MAPF komunitě se běžně využívají grafy ve formě 2D obdélníkové mřížky, které obsahují různé množství překážek. Proto se i my v této kapitole omezíme právě na ně. Agenti se budou pohybovat po políčkách mřížky, jejichž polohu lze zapsat pomocí souřadnic $x$ a $y$, jak je běžné pro body ve 2D prostoru. Nachází-li se agent na políčku se souřadnicemi $[x, y]$, může se v následujícím kroku posunout na políčka $[x + 1, y]$, $[x - 1, y]$, $[x, y + 1]$ a $[x, y - 1]$, pokud zde není překážka. Rovněž může také zůstat stát na původních souřadnicích $[x, y]$.

Na základě hustoty a rozmístění překážek lze grafy rozdělit do několika kategorií.

- **Prázdné mřížky $n \times n$** – nejjednodušší typ grafu. Snažíme se zde zpravidla o umístění co největšího počtu agentů a nalezení jejich cest v předepsaném čase. Hledání nejkratších cest jednotlivých agentů je velmi přímočaré, ale existuje velké množství potenciálních konfliktů mezi agenty. Zkoumáme zde především rychlost, s jakou se dokáží dané řešiče vypořádat s konflikty při plánování.

- **Bludiště** – grafy obsahující velké množství překážek. Ty mají souvislý charakter a značně omezují počet cest, které vedou k cíli. Na rozdíl od předchozího typu se může velmi lišit délka nejkratší přípustné cesty ze startovního vrcholu do cílového, oproti délce vzdušné čáry mezi těmito vrcholy.

- **Mřížka s náhodnými překážkami** – kombinace předchozích grafů. Základem je prázdná mřížka, ve které jsou náhodně odstraněny některé vrcholy. Agenti se vyhýbají nejen mezi sebou, ale i překážkám v prostředí.

- **Mapy z oblasti počítačových her** – plánování pohybu větší skupiny agentů najdeme v mnoha strategických a bojových počítačových hrách. Jedná se především o grafy se souvislou, avšak nepravidelnou průchozí plochou, kterou lemují překážky.

- **Mapy inspirované reálnými městy** – odpovídají zaměření na praktické problémy v oblasti řízení dopravy. Z hlediska robustnosti pak máme obvykle i dobrou představu o tom, s jakými zpožděními agentů můžeme počítat.
- **Skladiště** – grafy obsahující dlouhé úzké cesty, podél nichž jsou souvislé bloky překážek. Systém křížení cest je typicky pravoúhlý. Cílem je simulace prostředí skladů, ve kterých agenti přepravují zboží.

Všechny výše popsané typy grafů můžeme nalézt ve volně dostupné sadě MAPF benchmarků (viz Ben), uvedené v článku Stern a kol. (2019). I my budeme při experimentech čerpat právě odtud.

#### Počet agentů a výběr jejich lokace

Ve chvíli, kdy máme vybraný určitý typ prostředí, musíme určit, kolik agentů a jakým způsobem do něj umístíme. Pro námi používanou sadu benchmarků existují ke každému prostředí seznamy startovních a cílových pozic.

Postup testování na konkrétním grafu může v zásadě probíhat dvěma způsoby. V první variantě zvolíme počet agentů a díváme se na kvalitu obdržených výsledků. Druhou variantou je nalezení co nejvíce validních plánů v daném časovém limitu.

#### Pravděpodobnost zpoždění

V rámci rozšíření klasického MAPF problému uvažujeme v plánech jednotlivých agentů možnost opoždění se během cesty. Pro každého z agentů počítáme po celou dobu vykonávání plánu s pravděpodobností zpoždění rovnou konstantnímu číslu $p_z$ z intervalu $\langle 0, 1)$. Při implementaci postupujeme tak, že s pravděpodobností $p_z$ agent nevykoná akci přesunu (*move* akci), kterou měl dle plánu vykonat a zůstane stát na svém původním místě. Pro *wait* akce zpoždění neuvažujeme.

#### Pravděpodobnost bezkolizní exekuce plánu

Na rozdíl od klasického MAPF problému nás nezajímá pouze délka řešení, ale důležitá je především pravděpodobnost, s jakou při vykonávání plánu dojde ke kolizi mezi agenty. S danou pravděpodobností zpoždění simulujeme $n$ MAPF instancí a měříme, v kolika případech skončil plán úspěšně. Pravděpodobnost bezkolizní exekuce následně spočteme jako podíl úspěchů a počtu simulací.

## 7.1 Parametry experimentů

Experimenty provádíme postupně pro jednotlivé techniky. Testujeme na třech různých typech grafů s počtem 5, 10, 15 a 20 agentů. Není-li dále výslovně uvedeno jinak, pro daný graf a daný počet agentů vytvoříme celkem 500 MAPF instancí. Každá instance obsahuje náhodně vybrané startovní a cílové pozice agentů ze scénáře typu *random*, který je součástí dříve představených benchmarků. Startovní a cílové pozice jsou v rámci grafu totožné pro všechny testované metody robustnosti.

Pro zvolenou instanci hledáme plán splňující danou formu robustnosti. Pokud jsme řešení nalezli, provedeme exekuci plánu. V rámci exekuce plánů počítáme s pravděpodobností zpoždění agentů $p_z = 0{,}1$. Hodnotu průměrného makespanu a úspěšnost (podíl exekucí, které skončily bez kolize) počítáme ze všech vyřešených instancí. Při hledání řešení konkrétní instance omezujeme z časových důvodů dobu běhu algoritmu CBS nebo Picat řešiče na 20 sekund. Pokud do tohoto limitu neobdržíme požadovaný plán, označíme danou instanci jako nevyřešenou.

Experimenty provádíme v simulátoru, který je součástí elektronické přílohy A.2. Ovládání programu je uvedeno v uživatelské dokumentaci v příloze A.1. Měření času probíhalo na počítači s procesorem 3 GHz a 8 GB RAM.

## 7.2 Použité grafy a referenční hodnoty

Následuje bližší charakteristika grafů, které jsme vybrali pro experimenty. Snažili jsme se zahrnout grafy různých velikostí a typů s přihlédnutím na dostupný výpočetní výkon.

Referenční hodnoty jsou hodnoty průměrného makespanu a podílu bezkolizních exekucí klasického validního řešení nalezeného pomocí algoritmu CBS, ve kterém zakazujeme vrcholový konflikt a konflikt výměny vrcholů. Na obrázcích zachycujících jednotlivé grafy jsou přístupné vrcholy zobrazeny zeleně a překážky červeně.

### 7.2.1 Mřížka

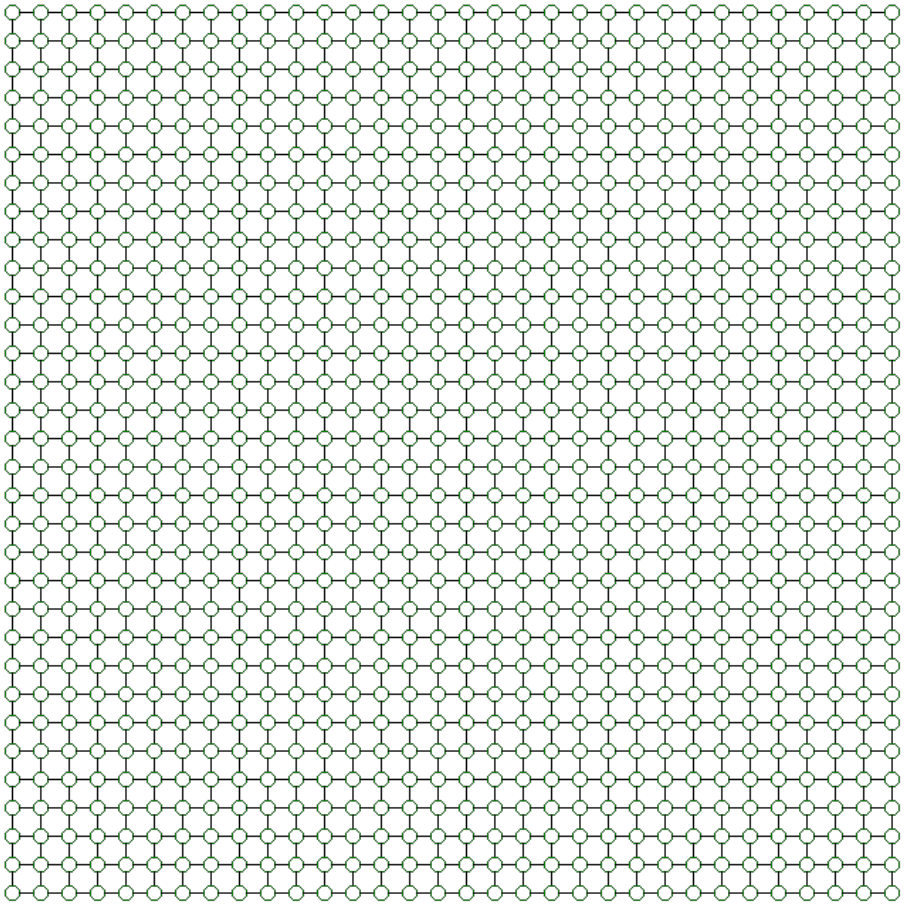
Pro testování jsme vybrali mřížku o rozměrech $32 \times 32$. Obsahuje celkem 1024 vrcholů, všechny jsou pro agenty přístupné. Na obrázku 7.1 vidíme její grafické znázornění. Referenční hodnoty uvádí tabulka 7.1.

| Počet agentů | Úspěšnost | Průměrný makespan |
|---|---|---|
| 5 | 0,88 | 37,97 |
| 10 | 0,53 | 42,92 |
| 15 | 0,23 | 45,7 |
| 20 | 0,08 | 47,16 |

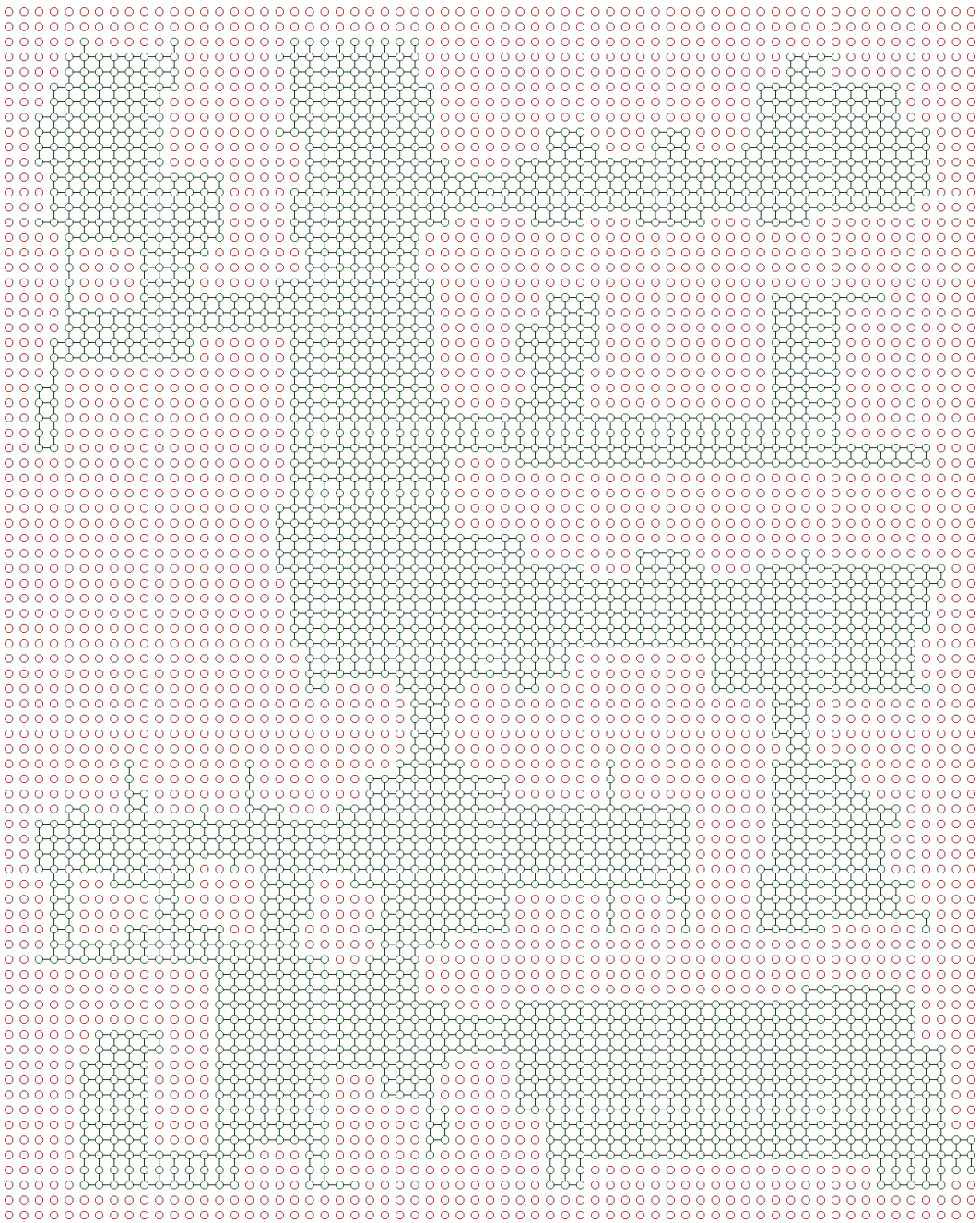
Tabulka 7.1: Naměřené referenční hodnoty pro graf Mřížka.

### 7.2.2 DAO mapa

Tento graf je vytvořený na základě mapy z počítačové hry Dragon Age: Origins. Jeho rozměry jsou $65 \times 81$ vrcholů, nicméně část, ve které se pohybují agenti, má nepravidelný tvar a skládá se z celkem 2445 vrcholů. Grafické znázornění je zachyceno na obrázku 7.2, naměřené referenční hodnoty vidíme v tabulce 7.2.



Obrázek 7.1: Graf – Mřížka.



Obrázek 7.2: Graf DAO mapa – mapa z počítačové hry Dragon Age: Origins.

| Počet agentů | Úspěšnost | Průměrný makespan |
|---|---|---|
| 5 | 0,76 | 96,78 |
| 10 | 0,3 | 108,22 |
| 15 | 0,09 | 114,62 |
| 20 | 0,01 | 117,64 |

Tabulka 7.2: Naměřené referenční hodnoty pro graf DAO mapa.

### 7.2.3 Skladiště

Jedná se o obdélníkový graf s rozměry $161 \times 63$. Obsahuje celkem 5699 vrcholů, na které mohou agenti vstoupit. Jeho podobu vidíme na obrázku 7.3. Tabulka 7.3 pak uvádí referenční hodnoty.

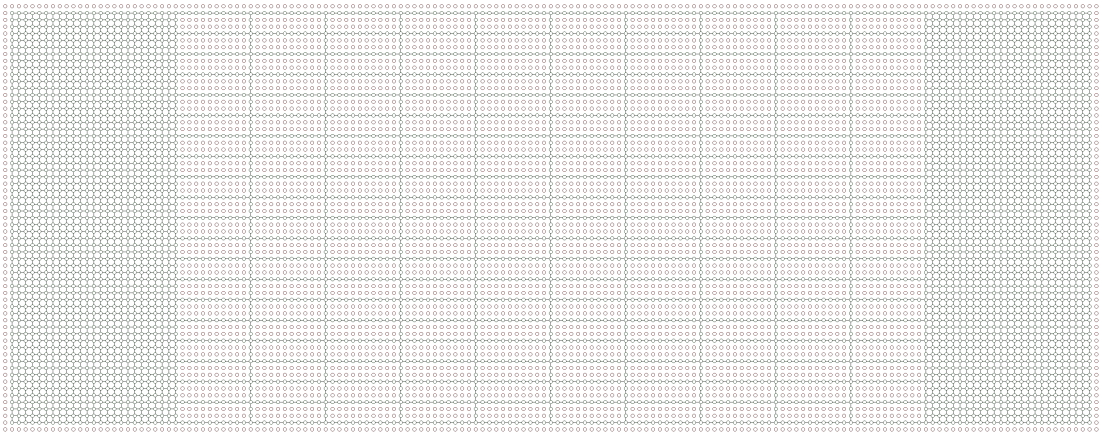
| Počet agentů | Úspěšnost | Průměrný makespan |
|---|---|---|
| 5 | 0,95 | 152,39 |
| 10 | 0,77 | 171,25 |
| 15 | 0,43 | 179,12 |
| 20 | 0,22 | 182,85 |

Tabulka 7.3: Naměřené referenční hodnoty pro graf Skladiště.

### 7.2.4 Shrnutí

Z výsledků experimentů nepřekvapivě vyčteme nárůst průměrné hodnoty makespanu jednak vzhledem ke vzrůstajícímu počtu agentů, jednak k velikosti grafu, na kterém testujeme.

U úspěšnosti bychom nejspíš očekávali opačný trend, protože s větším počtem vrcholů dostávají agenti v grafu více prostoru a měli by se na cestách potkávat méně. Nicméně nejnižší úspěšnost jsme zaznamenali v případě grafu DAO mapa. Z analýzy kolizí vyplývá, že tento jev je zapříčiněný nepravidelností mapy, ve které



Obrázek 7.3: Graf – Skladiště.

se nachází větší počet úzkých hrdel, přes která agenti projíždějí. V případě Mřížky se jedná o otevřenou mapu s rovnoměrným rozmístěním agentů. V grafu Skladiště mají agenti k dispozici dlouhé úzké cesty, ve kterých se příliš často nepotkávají – například je vyloučeno, aby vjeli do stejné uličky proti sobě, protože by vznikl konflikt vzájemné výměny vrcholů.

Počet vyřešených instancí z celkového počtu 500 v limitu 20 sekund na každou z nich je uveden v tabulce 7.4. Vidíme, že algoritmus CBS dokázal průměrně vyřešit přes 99 % instancí, v případě Mřížky byly vyřešeny všechny instance.

Navržený deklarativní model v jazyce Picat měl s nalezením řešení v daném limitu problém už pro instance s 5 agenty. V grafu DAO mapa vyřešil jen 10 instancí z 50 a v grafu Skladiště dokonce nevyřešil instanci žádnou. Úspěšný byl pouze na Mřížce. Navíc tento deklarativní model ve své 0-robustní variantě neřeší konflikt vzájemné výměny vrcholů a jak bylo zjištěno v práci Barták a kol. (2017), přidání příslušné podmínky pro kontrolu tohoto konfliktu představuje výrazný nárůst doby běhu. Z tohoto důvodu budeme v dalších námi navržených metodách robustnosti používat k nalezení (hlavního) plánu algoritmus CBS.

| | Vyřešených instancí | | |
|---|---|---|---|
| Počet agentů | Mřížka | DAO mapa | Skladiště |
| 5 | 500 | 500 | 500 |
| 10 | 500 | 500 | 499 |
| 15 | 500 | 498 | 493 |
| 20 | 500 | 478 | 490 |

Tabulka 7.4: Počty vyřešených instancí algoritmem CBS.

## 7.3 Výsledky existujících přístupů

Hodnoty naměřené pro klasické validní řešení MAPF uvádíme jako referenční v rámci předchozí sekce. Dále nás bude zajímat především $k$-robustnost, protože s tímto pojmem částečně pracujeme u definic alternativní a semi $k$-robustnosti.

### 7.3.1 *K*-robustnost

Tabulky 7.5, 7.6 a 7.7 uvádí výsledky experimentů pro parametr $k = 1$ a $k = 2$ ve vybraných testovacích grafech. Průměrný počet vyřešených instancí ve všech grafech zachycuje tabulka 7.8.

Plány jsme hledali pomocí upraveného CBS algoritmu. Konkrétně jsme použili implementaci vylepšeného $k$-robustního CBS. Tento postup byl výrazně rychlejší než použití Picat řešiče.

### 7.3.2 *P*-robustnost

V případě $p$-robustnosti by neměly prováděné experimenty velkou vypovídající hodnotu, neboť jejich výsledky by velmi závisely na nastavení parametrů řešiče hledajícího tyto druhy plánů. Chceme-li totiž najít plán, který splňuje parametry $p$-robustnosti, postupujeme následovně:

1. provedeme odhad zpoždění jednotlivých agentů,
2. s tímto odhadem simulujeme provádění plánu, přičemž plán měníme tak dlouho, dokud v rámci statistického testu (konkrétně se zde používá $Z$-test) nevyjde podíl bezkolizních exekucí větší nebo roven hodnotě $p$. Počet exekucí, které v rámci testu provádíme, může být buď fixní, nebo se dynamicky měnit za běhu algoritmu. Konkrétní volba této hodnoty má významný vliv na dobu hledání plánu.

Pokud v bodu 1 nastavíme správný odhad zpoždění (v našem případě činí 0,1), můžeme při experimentech očekávat úspěšnost na hranici parametru $p$. Pokud bychom odhad úmyslně podhodnotili, budeme se blížit úspěšnosti klasického validního plánu (pro odhadované zpoždění blížící se nule).

Co se týče bodu 2, probíhá hledání plánu velmi pomalu, takže by nastal problém s časovým limitem 20 sekund na instanci. Ve skutečnosti by nastal problém i pro daleko vyšší limit. Pro bližší představu uvedeme výsledky $p$-robustního přístupu z článku Atzmon a kol. (2019). V něm se prováděl experiment na grafu typu mřížka s rozměry $8 \times 8$ a s osmi náhodně rozmístěnými agenty. Testovaly se hodnoty $p = 0{,}7$ a $p = 0{,}9$, které se porovnávaly s klasickým plánem nalezeným pomocí algoritmu CBS.

Doba běhu nutná k nalezení 0,7-robustního plánu byla více jak 800krát delší oproti klasickému CBS plánu. Nalezení plánu pro $p = 0{,}9$ pak trvalo ještě přibližně 5krát déle. Hodnota účelové funkce $p$-robustních plánů rostla oproti základnímu řešení v řádu desítek procent, jednalo se však nejspíš o účelovou funkci SoC. Počet bezkolizních exekucí byl ve srovnání s klasickým CBS plánem zhruba dvojnásobný.

| | Úspěšnost | | Průměrný makespan | |
|---|---|---|---|---|
| Počet agentů | k=1 | k=2 | k=1 | k=2 |
| 5 | 0,96 | 0,98 | 38,32 | 38,07 |
| 10 | 0,75 | 0,87 | 43,17 | 43,18 |
| 15 | 0,45 | 0,71 | 45,87 | 45,95 |
| 20 | 0,23 | 0,52 | 47,4 | 47,6 |

Tabulka 7.5: Naměřené hodnoty $k$-robustnosti pro graf Mřížka.

| | Úspěšnost | | Průměrný makespan | |
|---|---|---|---|---|
| Počet agentů | k=1 | k=2 | k=1 | k=2 |
| 5 | 0,83 | 0,9 | 96,6 | 96,45 |
| 10 | 0,43 | 0,59 | 108,41 | 108,44 |
| 15 | 0,15 | 0,28 | 114,33 | 113,89 |
| 20 | 0,04 | 0,12 | 116,87 | 116,47 |

Tabulka 7.6: Naměřené hodnoty $k$-robustnosti pro graf DAO mapa.

| | Úspěšnost | | Průměrný makespan | |
|---|---|---|---|---|
| Počet agentů | k=1 | k=2 | k=1 | k=2 |
| 5 | 0,96 | 0,96 | 152,45 | 152,24 |
| 10 | 0,78 | 0,86 | 171,24 | 171,34 |
| 15 | 0,55 | 0,62 | 179,16 | 179,09 |
| 20 | 0,29 | 0,42 | 182,78 | 182,5 |

Tabulka 7.7: Naměřené hodnoty $k$-robustnosti pro graf Skladiště.

| | Průměrný počet vyřešených instancí | |
|---|---|---|
| Graf | k=1 | k=2 |
| Mřížka | 497,75 | 482,5 |
| DAO mapa | 474,25 | 438 |
| Skladiště | 498 | 489,75 |

Tabulka 7.8: Průměrný počet vyřešených instancí v grafech – $k$-robustnost.

### 7.3.3 Shrnutí

$K$-robustnost i $p$-robustnost dokáží podstatným způsobem zvýšit robustnost plánů oproti základnímu řešení. U $k$-robustnosti se zlepšení děje zejména na úkor času. Hodnota makespanu nijak dramaticky neroste, naopak zůstává téměř stejná jako u optimálního řešení. V některých případech pak dostáváme u 2-robustních plánů menší hodnotu průměrného makespanu než u plánů 1-robustních. Příčinou je velmi pravděpodobně menší počet vyřešených instancí, protože k vypršení časového limitu zpravidla dochází u dlouhých plánů s vyšší hodnotou účelové funkce. Tento jev pak zejména v grafu DAO mapa zapříčinil dokonce to, že hodnota účelové funkce je menší než v klasickém validním řešení.

Zajímavostí je, že 1-robustní řešič průměrně vyřešil v grafu Skladiště více instancí než klasický (0-robustní) řešič. Při hledání požadovaného plánu tak zřejmě převážil efekt větších omezení nad složitostí ho nalézt. Jinými slovy 1-zpožděné konflikty pomohly více omezit prostor řešení.

V případě $p$-robustnosti lze z uvedeného experimentu předpokládat, že kromě výrazného nárůstu doby běhu můžeme očekávat i určité zvýšení délky plánu.

## 7.4 Robustnost zajištěná alternativními plány + robustnost pomocí přidání *wait* akcí

V této sekci prezentujeme výsledky popsaných technik z kapitoly 4. Hlavní plány jsme generovali pomocí algoritmu CBS, jehož výsledek jsme následně upravili technikou zvýšení robustnosti pomocí přidání *wait* akcí, kterou rozebíráme v kapitole 5.

Do počtu vyřešených instancí se nyní, kromě časového limitu pro algoritmus CBS, přidává také samotná existence plánu splňujícího příslušnou formu robustnosti. V případě, kdy není možné vygenerovat požadovanou alternativní cestu (příkladem může být situace uvedená na obrázku 4.2) a daný robustní plán neexistuje, označíme tuto instanci jakou nevyřešenou.

### 7.4.1 Alternativní *k*-robustnost

Alternativně $k$-robustní plány jsme dále rozdělili na volné a striktní dle toho, jakým způsobem řeší konflikty mezi agenty na alternativních cestách. Výsledky experimentu pro volný přístup najdeme v tabulkách 7.9, 7.10 a 7.11, naměřené hodnoty striktního přístupu uvádíme v tabulkách 7.12, 7.13 a 7.14.

Jak jsme již zmínili v úvodu této sekce, jistých změn doznal i počet vyřešených instancí. Průměrné počty pro jednotlivé grafy zachycuje tabulka 7.15.

### 7.4.2 Semi *k*-robustnost

Naměřené hodnoty v rámci testů pro různé hodnoty parametru $k$ uvádíme v tabulkách 7.16 (pro graf Mřížka), 7.17 (pro graf DAO mapa) a 7.18 (pro graf Skladiště). Tabulka 7.19 zobrazuje počet vyřešených instancí pro jednotlivé grafy.

| | Úspěšnost | | | Průměrný makespan | | |
|---|---|---|---|---|---|---|
| Počet agentů | k=2 | k=3 | k=4 | k=2 | k=3 | k=4 |
| 5 | 0,97 | 0,98 | 0,99 | 38,05 | 38,16 | 38,2 |
| 10 | 0,8 | 0,86 | 0,87 | 43,16 | 43,45 | 43,62 |
| 15 | 0,58 | 0,71 | 0,72 | 45,94 | 46 | 46,48 |
| 20 | 0,39 | 0,52 | 0,54 | 47,54 | 47,71 | 48,09 |

Tabulka 7.9: Naměřené hodnoty alternativní $k$-robustnosti pro graf Mřížka – volný přístup.

| | Úspěšnost | | | Průměrný makespan | | |
|---|---|---|---|---|---|---|
| Počet agentů | k=2 | k=3 | k=4 | k=2 | k=3 | k=4 |
| 5 | 0,85 | 0,84 | 0,92 | 96,65 | 96,65 | 96,87 |
| 10 | 0,47 | 0,58 | 0,67 | 108,69 | 109,06 | 109,43 |
| 15 | 0,17 | 0,24 | 0,36 | 115,12 | 115,56 | 116,16 |
| 20 | 0,05 | 0,09 | 0,13 | 118,3 | 119,33 | 120,4 |

Tabulka 7.10: Naměřené hodnoty alternativní $k$-robustnosti pro graf DAO mapa – volný přístup.

| | Úspěšnost | | | Průměrný makespan | | |
|---|---|---|---|---|---|---|
| Počet agentů | k=2 | k=3 | k=4 | k=2 | k=3 | k=4 |
| 5 | 0,95 | 0,96 | 0,96 | 152,34 | 152,24 | 152,16 |
| 10 | 0,83 | 0,82 | 0,88 | 171,19 | 171,23 | 170,87 |
| 15 | 0,55 | 0,61 | 0,64 | 179,29 | 179,41 | 179,07 |
| 20 | 0,31 | 0,38 | 0,44 | 182,84 | 182,73 | 183 |

Tabulka 7.11: Naměřené hodnoty alternativní $k$-robustnosti pro graf Skladiště – volný přístup.

| | Úspěšnost | | | Průměrný makespan | | |
|---|---|---|---|---|---|---|
| Počet agentů | k=2 | k=3 | k=4 | k=2 | k=3 | k=4 |
| 5 | 0,96 | 0,98 | 0,99 | 38,18 | 38,15 | 38,11 |
| 10 | 0,81 | 0,87 | 0,92 | 43,08 | 43,25 | 43,23 |
| 15 | 0,58 | 0,76 | 0,76 | 46,01 | 46,15 | 46,15 |
| 20 | 0,37 | 0,54 | 0,66 | 47,61 | 47,38 | 47,48 |

Tabulka 7.12: Naměřené hodnoty alternativní $k$-robustnosti pro graf Mřížka – striktní přístup.

| | Úspěšnost | | | Průměrný makespan | | |
|---|---|---|---|---|---|---|
| Počet agentů | k=2 | k=3 | k=4 | k=2 | k=3 | k=4 |
| 5 | 0,85 | 0,86 | 0,89 | 96,65 | 96,56 | 96,95 |
| 10 | 0,48 | 0,57 | 0,68 | 108,77 | 108,76 | 109 |
| 15 | 0,18 | 0,33 | 0,43 | 114,8 | 115,02 | 114,99 |
| 20 | 0,07 | 0,16 | 0,2 | 118,15 | 117,11 | 117,87 |

Tabulka 7.13: Naměřené hodnoty alternativní $k$-robustnosti pro graf DAO mapa – striktní přístup.

| | Úspěšnost | | | Průměrný makespan | | |
|---|---|---|---|---|---|---|
| Počet agentů | k=2 | k=3 | k=4 | k=2 | k=3 | k=4 |
| 5 | 0,95 | 0,96 | 0,96 | 152,34 | 152,24 | 152,17 |
| 10 | 0,83 | 0,82 | 0,88 | 171,19 | 171,23 | 170,87 |
| 15 | 0,55 | 0,6 | 0,64 | 179,29 | 179,42 | 179,02 |
| 20 | 0,29 | 0,37 | 0,44 | 182,99 | 182,85 | 183,07 |

Tabulka 7.14: Naměřené hodnoty alternativní $k$-robustnosti pro graf Skladiště – striktní přístup.

| | Průměrný počet vyřešených instancí | | | | | |
|---|---|---|---|---|---|---|
| | Volný přístup | | | Striktní přístup | | |
| Graf | k=2 | k=3 | k=4 | k=2 | k=3 | k=4 |
| Mřížka | 500 | 500 | 500 | 473,5 | 455 | 427,75 |
| DAO mapa | 492,5 | 491,75 | 491,75 | 453 | 416,75 | 377,75 |
| Skladiště | 495,25 | 495,75 | 495 | 495,25 | 495,25 | 494,25 |

Tabulka 7.15: Průměrný počet vyřešených instancí pro jednotlivé grafy – alternativní $k$-robustnost.

| | Úspěšnost | | | Průměrný makespan | | |
|---|---|---|---|---|---|---|
| Počet agentů | k=2 | k=3 | k=4 | k=2 | k=3 | k=4 |
| 5 | 0,95 | 0,95 | 0,97 | 38,21 | 38,19 | 38,13 |
| 10 | 0,74 | 0,81 | 0,83 | 43,07 | 43,08 | 43,14 |
| 15 | 0,51 | 0,6 | 0,67 | 45,7 | 45,78 | 46,06 |
| 20 | 0,27 | 0,4 | 0,49 | 47,37 | 47,56 | 47,47 |

Tabulka 7.16: Naměřené hodnoty semi $k$-robustnosti pro graf Mřížka.

| | Úspěšnost | | | Průměrný makespan | | |
|---|---|---|---|---|---|---|
| Počet agentů | k=2 | k=3 | k=4 | k=2 | k=3 | k=4 |
| 5 | 0,82 | 0,83 | 0,87 | 96,57 | 96,71 | 96,83 |
| 10 | 0,43 | 0,46 | 0,52 | 108,75 | 108,38 | 108,68 |
| 15 | 0,14 | 0,19 | 0,2 | 114,56 | 114,61 | 114,91 |
| 20 | 0,04 | 0,06 | 0,07 | 117,96 | 117,58 | 117,66 |

Tabulka 7.17: Naměřené hodnoty semi $k$-robustnosti pro graf DAO mapa.

| | Úspěšnost | | | Průměrný makespan | | |
|---|---|---|---|---|---|---|
| Počet agentů | k=2 | k=3 | k=4 | k=2 | k=3 | k=4 |
| 5 | 0,95 | 0,96 | 0,96 | 152,35 | 152,33 | 152,37 |
| 10 | 0,81 | 0,85 | 0,84 | 171,41 | 171,27 | 171,23 |
| 15 | 0,55 | 0,58 | 0,64 | 179,21 | 179,33 | 179,18 |
| 20 | 0,32 | 0,36 | 0,4 | 183,11 | 183,27 | 182,87 |

Tabulka 7.18: Naměřené hodnoty semi $k$-robustnosti pro graf Skladiště.

| | Průměrný počet vyřešených instancí | | |
|---|---|---|---|
| Graf | k=2 | k=3 | k=4 |
| Mřížka | 500 | 500 | 500 |
| DAO mapa | 471,5 | 465,5 | 459,5 |
| Skladiště | 493,5 | 493,25 | 490,5 |

Tabulka 7.19: Průměrný počet vyřešených instancí pro jednotlivé grafy – semi $k$-robustnost.

### 7.4.3 Vliv robustnosti pomocí přidávání *wait akcí*

V této části stručně rozebereme, jakých výsledků dosahuje samotné přidávání *wait* akcí a jaký má vliv na zvýšení robustnosti u plánů s alternativami. Jak víme, tak tato forma robustnosti nezvyšuje makespan plánu a jeho porovnávání by tak bylo zbytečné.

Na všech třech grafech jsem provedli experiment porovnávající úspěšnost technik alternativní 3-robustnost*, semi 3-robustnost* (u obou tentokrát bez úpravy hlavního řešení pomocí *wait* akcí, což označujeme symbolem *) a robustnost pomocí přidávání *wait* akcí pro hodnotu minimálního bezpečného intervalu $b = 3$. V tabulkách poslední jmenovanou metodu označujeme jako *wait*-3. Pro hledání validního řešení MAPF jsme použili algoritmus CBS.

Naměřené hodnoty pro graf Mřížka najdeme v tabulce 7.20. Výsledky pro graf DAO mapa pak zachycuje tabulka 7.21 a v tabulce 7.22 vidíme, jak dopadlo porovnání v grafu Skladiště.

### 7.4.4 Shrnutí

Navržené techniky alternativní $k$-robustnosti a semi $k$-robustnosti jsme vyzkoušeli s různými hodnotami parametrů na různých grafech.

Z naměřených hodnot obecně vyplývá (pro všechny varianty), že s rostoucí hodnotou parametru $k$ roste podíl úspěšnosti exekuce plánů. Patrný je zejména při vyšším počtu agentů, kde zvýšení parametru $k$ z hodnoty 2 na hodnotu 4 znamená i stoprocentní nárůst počtu bezkolizních provedení. Naopak průměrná hodnota účelové funkce roste se zvýšením parametru $k$ jen velmi málo, a sice v nízkých jednotkách procent. Jedná se tedy o velmi pozitivní zjištění.

Zároveň počet vyřešených instancí klesá jen nepatrně, opět se jedná o rozdíl jednotek procent mezi hodnotou parametru $k = 2$ a $k = 4$. Oproti $k$-robustnosti je tak tento vliv na vývoj délky účelové funkce zanedbatelný. Jedinou výjimkou je striktní alternativní $k$-robustnost, kde pozorujeme strmější pokles počtu vyřešených instancí. Je to dáno vyšší výpočetní složitostí, protože zde vlastně řešíme v rámci jedné instance dva MAPF problémy. Se zvyšujícím se parametrem $k$ samozřejmě stoupá počet alternativ, které je nutné vyhledat. Zatímco v případě volných přístupů se pouze odpovídajícím způsobem zvýší počet běhů algoritmu A* pro jednoho agenta, ve striktním přístupu se vytvoří MAPF problém s více agenty, kde se všichni musí koordinovat najednou.

#### Alternativní $k$-robustnost

Porovnáme-li striktní a volnou variantu alternativní $k$-robustnosti, pak zjistíme, že při zvoleném zpoždění $p_z = 0{,}1$ se požadavek na omezení alternativních konfliktů nejeví jako zásadně přínosný. Jeho vliv na zvýšení robustnosti je diskutabilní a převažuje negativní efekt v podobě značného prodloužení času nutného k vyřešení MAPF problému, který se projevil ve statistice počtu vyřešených instancí. Dá se předpokládat, že instance, jejichž řešení trvá déle a nevejdou se ve striktním přístupu do limitu 20 sekund, budou více náchylnější na výskyt kolizí.

V rámci porovnání volného a striktního přístupu jsme provedli ještě pokus zkoumající vliv velikosti zpoždění. Naměřili jsme, že se vzrůstající pravděpodobností zpoždění se úspěšnost obou přístupů mírně vychyluje ve prospěch striktního,

| | Úspěšnost | | |
|---|---|---|---|
| Počet agentů | *wait*-3 | alternativní 3-robustnost* | semi 3-robustnost* |
| 5 | 0,94 | 0,96 | 0,91 |
| 10 | 0,69 | 0,82 | 0,73 |
| 15 | 0,38 | 0,59 | 0,46 |
| 20 | 0,21 | 0,37 | 0,23 |

Tabulka 7.20: Vliv zvýšení robustnosti pomocí přidávání *wait* akcí – graf Mřížka.

| | Úspěšnost | | |
|---|---|---|---|
| Počet agentů | *wait*-3 | alternativní 3-robustnost* | semi 3-robustnost* |
| 5 | 0,83 | 0,84 | 0,77 |
| 10 | 0,43 | 0,52 | 0,32 |
| 15 | 0,16 | 0,17 | 0,1 |
| 20 | 0,03 | 0,05 | 0,02 |

Tabulka 7.21: Vliv zvýšení robustnosti pomocí přidávání *wait* akcí – graf DAO mapa.

| | Úspěšnost | | |
|---|---|---|---|
| Počet agentů | *wait*-3 | alternativní 3-robustnost* | semi 3-robustnost* |
| 5 | 0,89 | 0,96 | 0,93 |
| 10 | 0,7 | 0,81 | 0,76 |
| 15 | 0,43 | 0,56 | 0,5 |
| 20 | 0,18 | 0,3 | 0,21 |

Tabulka 7.22: Vliv zvýšení robustnosti pomocí přidávání *wait* akcí – graf Skladiště.

nejedná se však o žádný markantní rozdíl. Výsledný graf vidíme na obrázku 7.4.

### Semi *k*-robustnost

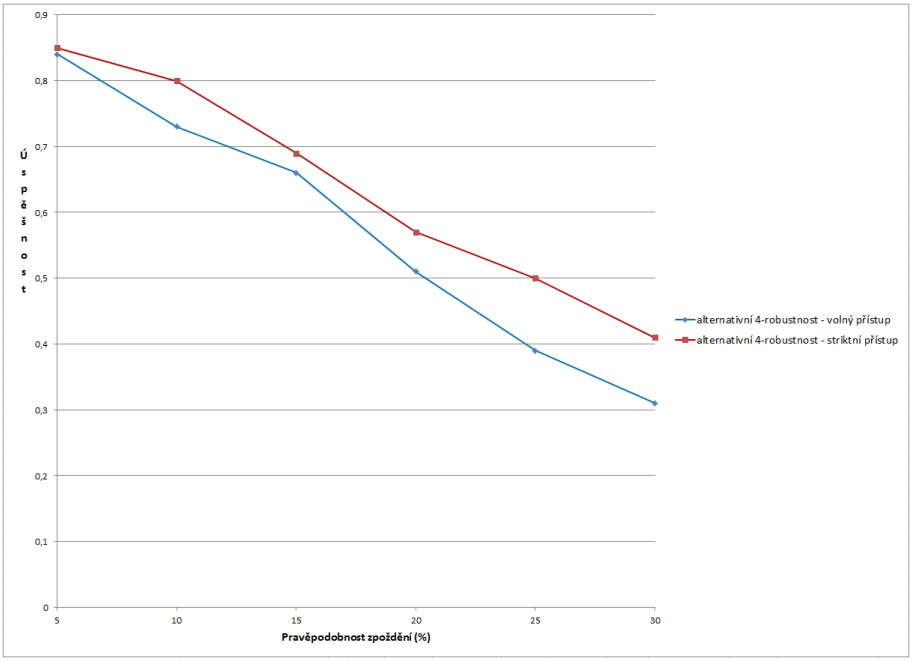
Semi $k$-robustnost vykazuje podobné charakteristiky jako volný přístup alternativní $k$-robustnosti. V obou případech se jedná o rychlé algoritmy zvyšující robustnost plánu za cenu nepatrného zvyšování jeho ceny. Při stejné hodnotě parametru $k$ ovšem z hlediska úspěšnosti dopadá o něco lépe alternativní $k$-robustnost. Semi $k$-robustnost se ale může pyšnit nepatrně menším průměrným makespanem.

Viditelným rozdílem je také o něco menší počet vyřešených instancí semi $k$-robustnosti v případě grafu DAO mapa. Tento jev nebyl způsoben selháním v důsledku nedostatku času. Alternativní cestu se nepodařilo najít, protože v danou chvíli taková neexistovala. Příčinou je strategie výběru větvících vrcholů. Na druhou stranu při konstrukci semi $k$-robustních plánů obecně vzniká více alternativních cest, což jsme ověřili v tabulce 7.23).

### Robustnost pomocí přidání *wait* akcí

Z výsledků, které jsme zjistili, se ukazuje, že tento algoritmus samotný dokáže nezanedbatelným způsobem zvýšit robustnost klasického validního řešení.

Rovněž se ukázalo, že velmi dobře funguje spolupráce s alternativní/semi $k$-robustností. Řešení vzniklá kombinací těchto metod mají smysl, protože se obě dvě na celkovém výsledku podílí nemalou měrou. V několika málo případech byla dokonce úspěšnost kombinace obou metod větší než součet úspěšností jednotlivých metod zvlášť.




Obrázek 7.4: Pokles úspěšnosti v závislosti na míře zpoždění v grafu Mřížka v instancích s 15 agenty.

| | Průměrný počet alternativních cest | |
|---|---|---|
| Graf | alternativní 4-robustnost | semi 4-robustnost |
| Mřížka | 16,88 | 41,74 |
| DAO mapa | 51,86 | 120,63 |
| Skladiště | 9,16 | 19,39 |

Tabulka 7.23: Průměrný počet přidaných alternativ v instancích s 20 agenty.

## 7.5 Robustnost pomocí změny rychlosti agentů

V případě techniky $min/max$ robustnosti, popsané v kapitole 6, jsme testovali se třemi různými dvojicemi parametrů $min$ a $max$. Číselné vyjádření této dvojice udává v každé z tabulek písmeno $r$. Naměřené hodnoty pro graf Mřížka najdeme v tabulce 7.24, výsledky na grafu DAO mapa zobrazuje tabulka 7.25 a výsledky z grafu Skladiště jsou uvedeny v tabulce 7.26.

Při hledání základního validního plánu, který slouží jako vstup pro tvorbu $min/max$ robustního plánu používáme algoritmus CBS. Počet vyřešených instancí tedy koresponduje s hodnotami, které jsou uvedeny v tabulce 7.4 u referenčního řešení.

### 7.5.1 Shrnutí

Vidíme, že implementovaná technika má velký vliv na zvýšení robustnosti v každém typu prostředí při všech testovaných dvojicích hodnot $min$ a $max$. Průměrný makespan pak zcela očekávaně vzrostl oproti makespanu klasického validního plánu zhruba v poměru daném zlomkem $\frac{max}{min}$.

Lze konstatovat, že pokud nám nevadí určité zvýšení délky plánu a můžeme si dovolit provést jeho zjemnění, je tato metoda skvělou volbou pro zajištění bezkolizního provedení. Čím větší zjemnění (parametr $max$) je možné provést, tím lépe. Nicméně jak bylo experimentálně ověřeno, už při trisekci intervalu jsou výsledky nadmíru uspokojivé.

| | Úspěšnost | | | Průměrný makespan | | |
|---|---|---|---|---|---|---|
| Počet agentů | r=2/3 | r=3/4 | r=3/5 | r=2/3 | r=3/4 | r=3/5 |
| 5 | 0,99 | 1 | 1 | 51,53 | 45,6 | 56,98 |
| 10 | 0,95 | 0,99 | 1 | 58,22 | 51,5 | 64,3 |
| 15 | 0,91 | 0,96 | 0,99 | 61,86 | 54,72 | 68,21 |
| 20 | 0,87 | 0,93 | 0,98 | 63,95 | 56,59 | 70,44 |

Tabulka 7.24: Naměřené hodnoty *min*/*max* robustnosti pro graf Mřížka.

| | Úspěšnost | | | Průměrný makespan | | |
|---|---|---|---|---|---|---|
| Počet agentů | r=2/3 | r=3/4 | r=3/5 | r=2/3 | r=3/4 | r=3/5 |
| 5 | 0,98 | 0,99 | 1 | 130,37 | 115,62 | 144,49 |
| 10 | 0,93 | 0,97 | 0,99 | 146,38 | 129,95 | 162,2 |
| 15 | 0,82 | 0,9 | 0,99 | 156,04 | 137,58 | 171,15 |
| 20 | 0,72 | 0,83 | 0,97 | 160,2 | 141,12 | 175,68 |

Tabulka 7.25: Naměřené hodnoty *min*/*max* robustnosti pro graf DAO mapa.

| | Úspěšnost | | | Průměrný makespan | | |
|---|---|---|---|---|---|---|
| Počet agentů | r=2/3 | r=3/4 | r=3/5 | r=2/3 | r=3/4 | r=3/5 |
| 5 | 0,99 | 1 | 1 | 205,4 | 182,37 | 227,98 |
| 10 | 0,98 | 0,99 | 1 | 230,92 | 204,79 | 256,08 |
| 15 | 0,95 | 0,97 | 1 | 242,24 | 215,16 | 268,32 |
| 20 | 0,89 | 0,93 | 0,98 | 247,93 | 219,38 | 273,66 |

Tabulka 7.26: Naměřené hodnoty *min*/*max* robustnosti pro graf Skladiště.

## 7.6 Vzájemné porovnání všech testovaných přístupů

Doposud jsme se zaměřovali na jednotlivé přístupy spíše samostatně s cílem zaznamenat přesné výsledky naměřených veličin a analyzovat jejich chování pro různé hodnoty parametrů. Nyní se přesuneme ke vzájemnému srovnání vybraných technik. Naším cílem bude zejména graficky znázornit trendy chování v různých aspektech. K testování vybereme z představených technik vždy jednoho zástupce, konkrétně 3/4 robustnost, alternativní 4-robustnost a semi 4-robustnost a srovnáme je s 1-robustním plánem, 2-robustním plánem a klasickým validním řešením.

### Úspěšnost

První věcí, kterou testujeme je vývoj podílu bezkolizních instancí v závislosti na počtu agentů. Pro daný počet agentů jsme vytvořili vždy 100 náhodných instancí, pravděpodobnost zpoždění činí 10 %. Agenty tentokrát přidáváme po jednom. Výsledné grafy vidíme na obrázcích 7.5 (Mřížka), 7.6 (DAO mapa) a 7.7 (Skladiště).

### Úspěšnost vs. cena

Dalším zajímavým aspektem k porovnání se zdá být poměr mezi úspěšností a délkou plánu (cenou). Na obrázku 7.8 vidíme bodový graf pro Mřížku, na obrázku 7.9 bodový graf pro DAO mapu a na obrázku 7.10 najdeme naměřené hodnoty pro Skladiště.

Při testování jsme provedli celkem 20 různých pokusů. V každém z nich jsme spočítali průměrné hodnoty na 100 náhodně vybraných instancích s 15 agenty při pravděpodobnosti zpoždění $p_z = 0{,}1$. Pro každou techniku jsme tedy dostali celkem 20 hodnot, které jsou zobrazeny stejnou barvou jako body ve 2D prostoru.

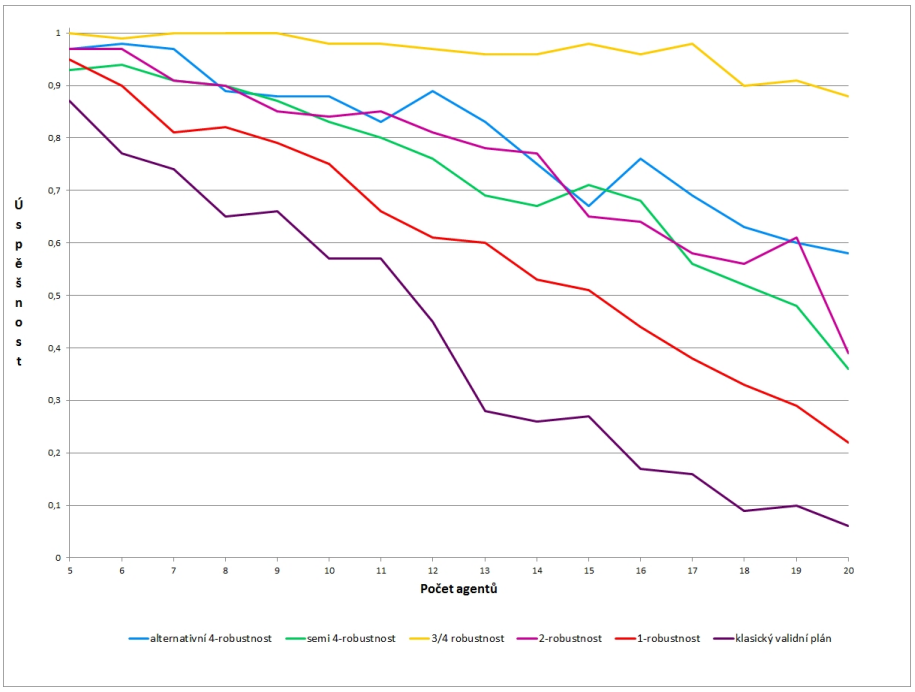
Abychom mohli provést názorné porovnání, nanášíme na osu $x$ tzv. nárůst délky plánu $n_\pi$. Ten spočítáme dle vzorce $n_\pi = 1 - \frac{makespan_{opt}}{makespan_\pi}$, kde $makespan_{opt}$ představuje makespan optimálního klasického validního plánu, pokud by nenastalo žádné zpoždění agentů. Tím docílíme, že na obou osách máme hodnoty mezi 0 a 1, přičemž daná technika je tím lepší, čím více leží dané hodnoty vlevo a nahoře.

### Rychlost

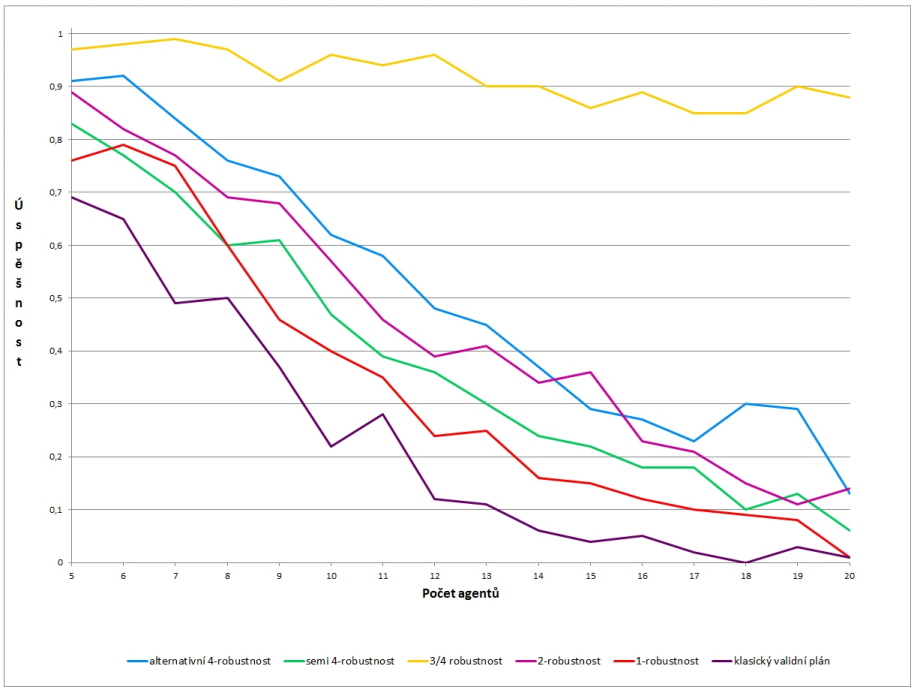
Měření rychlosti, s jakou dokážeme nalézt plány splňující podmínku robustnosti, jsme provedli dvěma způsoby.

V prvním z nich jsme zkoumali počet vyřešených náhodných instancí za dobu 30 sekund s fixním počtem deseti náhodně rozmístěných agentů. Pokus jsme provedli celkem 15krát a jako výsledek jsme vzali hodnotu mediánu počtu vyřešených instancí. Startovní a cílové vrcholy agentů byly vybírány různě v rámci pokusů, ale zůstávaly stejné pro každou testovanou robustní techniku. Na obrázku 7.11 vidíme medián počtu vyřešených instancí v grafu Mřížka s deseti agenty a na obrázcích 7.12, resp. 7.13 je zobrazeno totéž pro graf DAO mapa, resp. Skladiště.

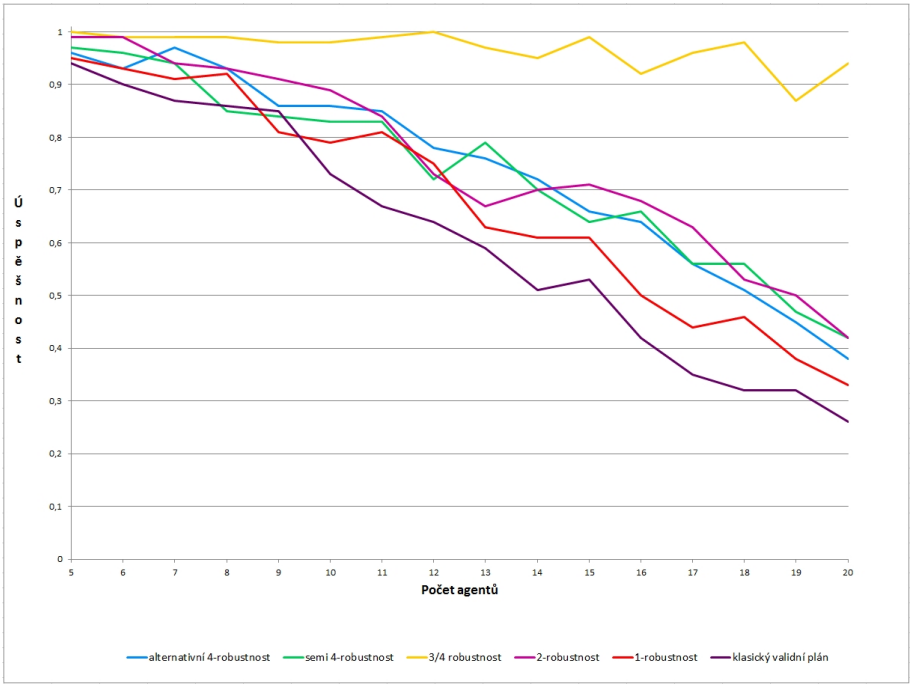
Druhý způsob testování spočíval v měření maximálního počtu agentů, pro které dokázal daný řešič najít plán s požadovanou robustností. Časový limit byl



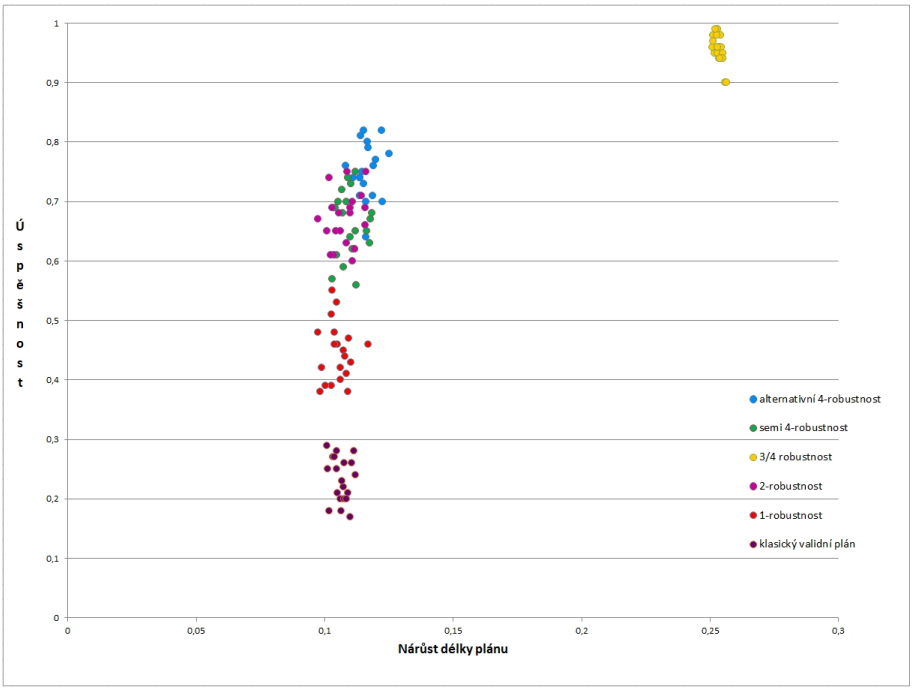

Obrázek 7.5: Porovnání úspěšnosti jednotlivých technik – graf Mřížka.



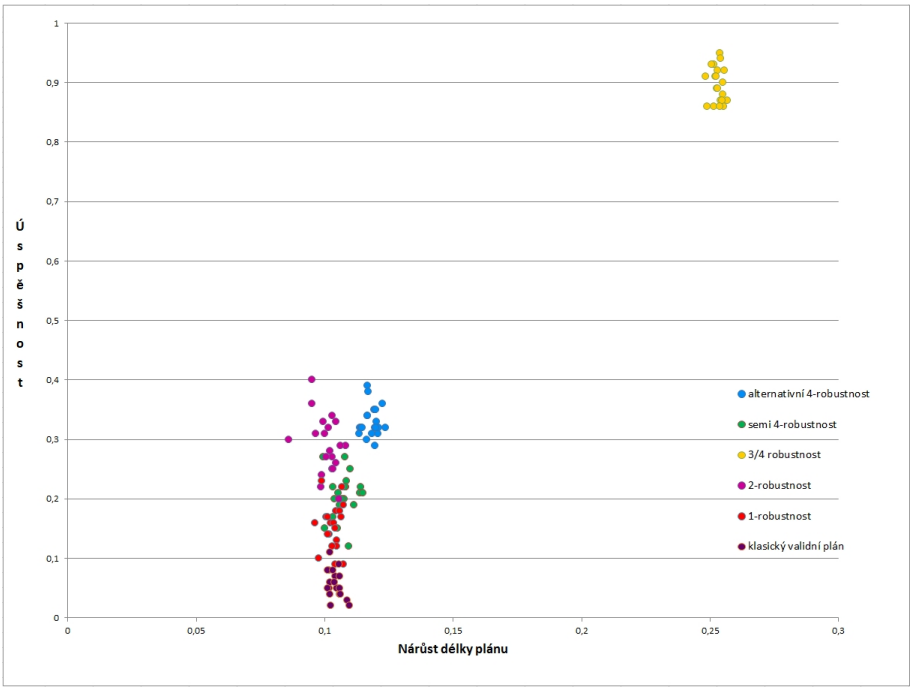

Obrázek 7.6: Porovnání úspěšnosti jednotlivých technik – graf DAO mapa.



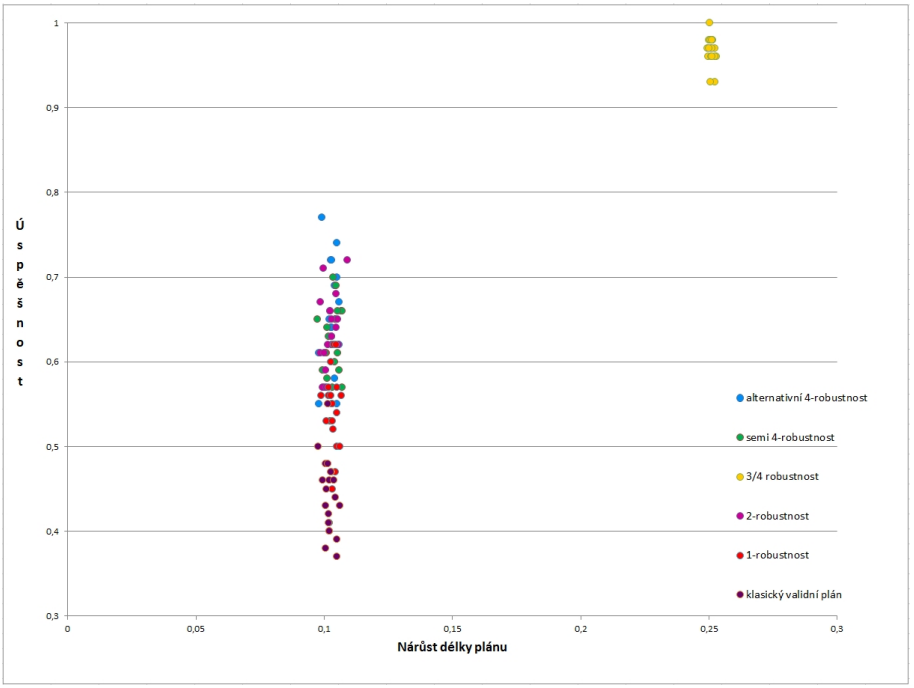

Obrázek 7.7: Porovnání úspěšnosti jednotlivých technik – graf Skladiště.




Obrázek 7.8: Porovnání úspěšnost vs. cena – graf Mřížka.




Obrázek 7.9: Porovnání úspěšnost vs. cena – graf DAO mapa.




Obrázek 7.10: Porovnání úspěšnost vs. cena – graf Skladiště.

opět 30 sekund. Začali jsme se dvěma agenty, přičemž po každém úspěšném vyřešení instance se počet agentů zvýšil o jedna. Celý test jsme opakovali 15krát a jako výsledek vzali medián. Na obrázku 7.14 uvádíme výsledky pro graf Mřížka. Dále jsme testovali na grafu DAO mapa (obrázek 7.15) a Skladiště, u kterého jsme opět dostali překvapivý výsledek. Bližší popis viz obrázek 7.16.

Do časového limitu zahrnujeme kromě nalezení požadovaného plánu i jeho exekuci, abychom postihli čas, který potřebuje *min/max* robustnost k sestavení zjemněného plánu.

### 7.6.1 Shrnutí

#### Úspěšnost

Z hlediska úspěšnosti vychází jednoznačně nejlépe 3/4 robustnost. Jedná se o odlišný přístup oproti ostatním, který vykazuje velkou odolnost vůči zpožděním, a to i se zvyšujícím se počtem agentů v grafu. Z metod založených na alternativních plánech vychází ze srovnání lépe alternativní 4-robustnost. Z naměřených hodnot se zdá, že dosahuje o něco lepších výsledků než 2-robustnost. V případě semi 4-robustnosti jsme naměřili hodnoty lepší než v případě 1-robustnosti. Na grafech Mřížka a Skladiště pak vychází srovnatelně s 2-robustností, ale na grafu DAO mapa je horší. Nejhorší úspěšnosti logicky dosahuje klasický validní plán. Rozdíl mezi ním a robustními přístupy byl patrnější na grafech s méně vrcholy. Z hlediska grafů jsou rozdíly nejvíce patrné na Mřížce, naopak nejblíže k sobě křivky mají v grafu Skladiště.

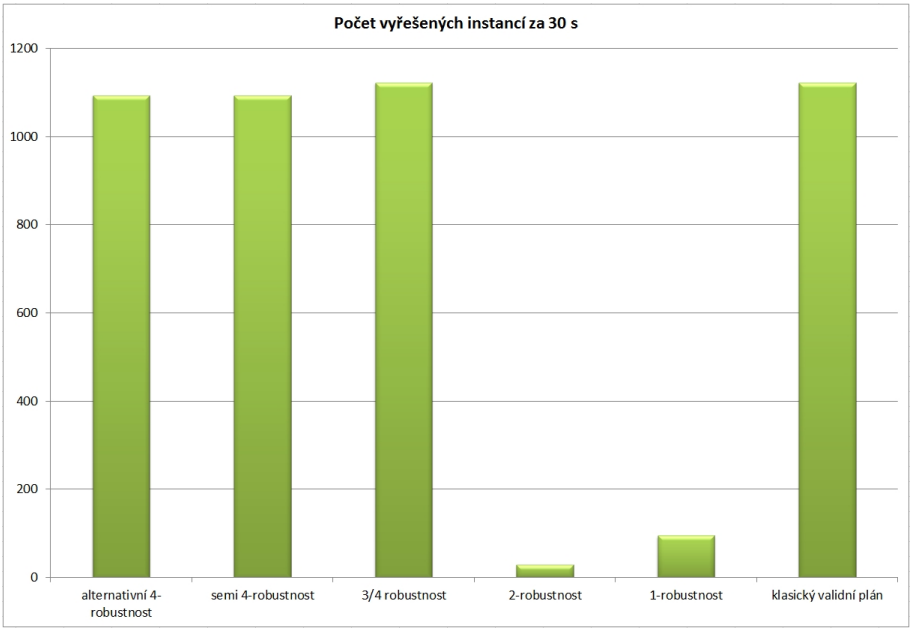
#### Úspěšnost vs. délka plánu

Porovnáváme-li úspěšnost versus délka plánu, pak v případě grafu Mřížka vidíme 4 clustery. Nejhorších výsledků dosáhl klasický validní plán, dále následoval 1-robustní plán. Do třetího clusteru bychom zařadili semi 4-robustnost, alternativní 4-robustnost a 2-robustnost. Čtvrtý cluster tvoří 3/4 robustnost. Poslední dva jmenované nejsou přímo porovnatelné, neboť v jednom jsou hodnoty s lepším makespanem a ve druhém hodnoty s lepší úspěšností.

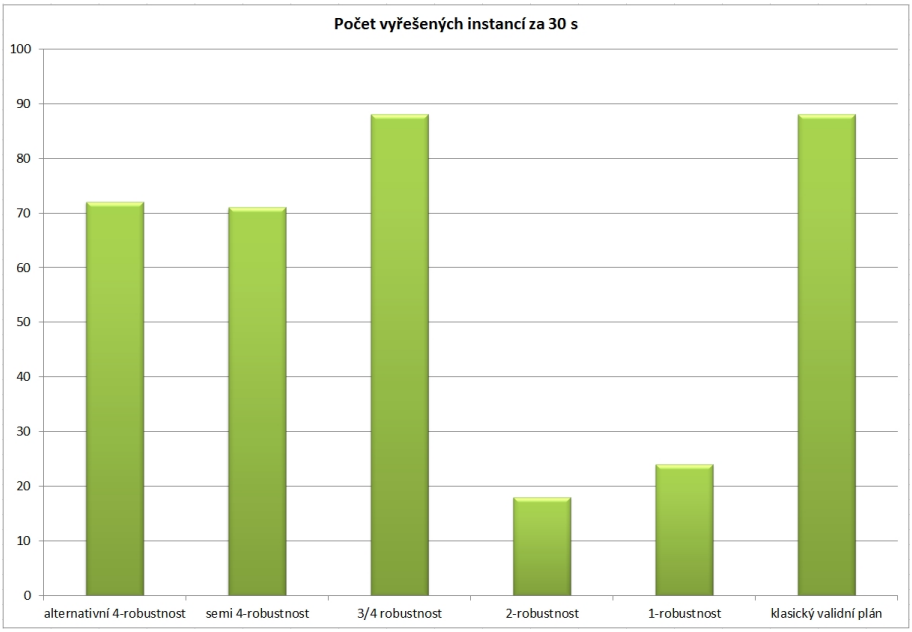
V případě grafu DAO mapa nám výsledky více splynuly. Zcela osamocena opět zůstala 3/4 robustnost, poměrně oddělené jsou hodnoty alternativní 4-robustnosti a 2-robustnosti. První uvedená má lepší úspěšnost, druhá zase kvalitu plánů. Následuje semi 4-robustnost a kousek pod ní se nachází 1-robustnost.

Nejméně přehledné jsou výsledky z grafu Skladiště. Zde jsme se nakonec pro lepší přehlednost rozhodli osu $x$ více natáhnout, takže její hodnoty začínají až od čísla 0,3. Nehledě na tuto operaci mají jednotlivé výsledky v rámci robustních metod největší rozptýlení na ose $y$ a naopak malé rozdíly můžeme vidět na ose $x$. Kromě osamocené 3/4 robustnosti pak vypadá další pořadí zhruba podobně jako u předchozích dvou grafů, ač bez rozlišitelných clusterů.

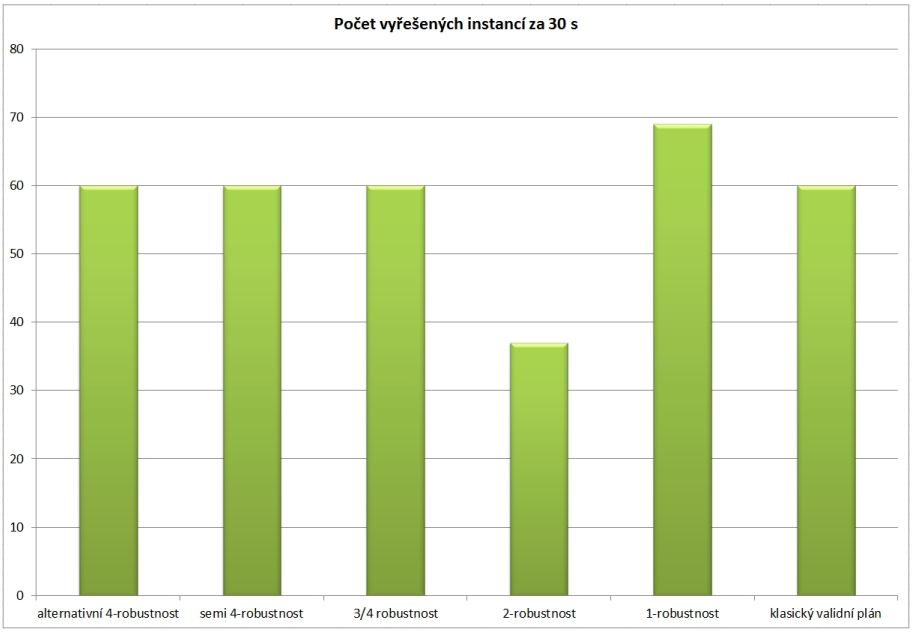
Mimochodem není náhoda, že ve všech třech grafech vidíme u většiny metod nárůst délky plánu na hodnotě těsně nad 0,1. Ten je způsoben zpožděním, které jsme nastavili na 10 %. Jedině u 3/4 robustnosti můžeme pozorovat nárůst délky plánu těsně nad hranicí 0,25. To znamená, že podíl délek plánů optimálního řešení (bez zpoždění) a 3/4 robustního řešení vychází okolo 0,75. Celkový nárůst délky proto činí $1/0{,}75 = 4/3$, což je předpokládaný nárůst u 3/4 robustního plánu.



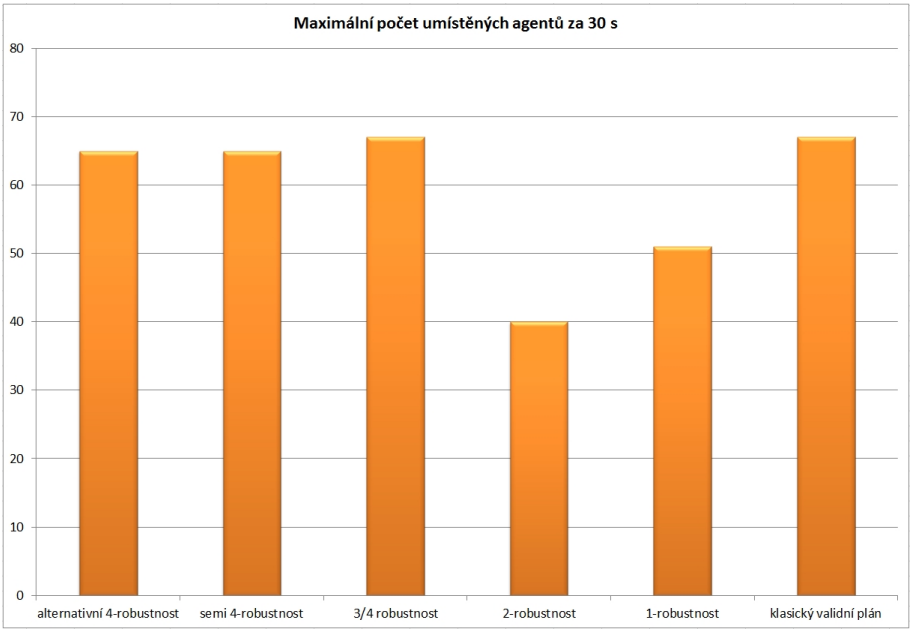

Obrázek 7.11: Medián počtu vyřešených instancí s 10 agenty za 30 sekund – graf Mřížka.



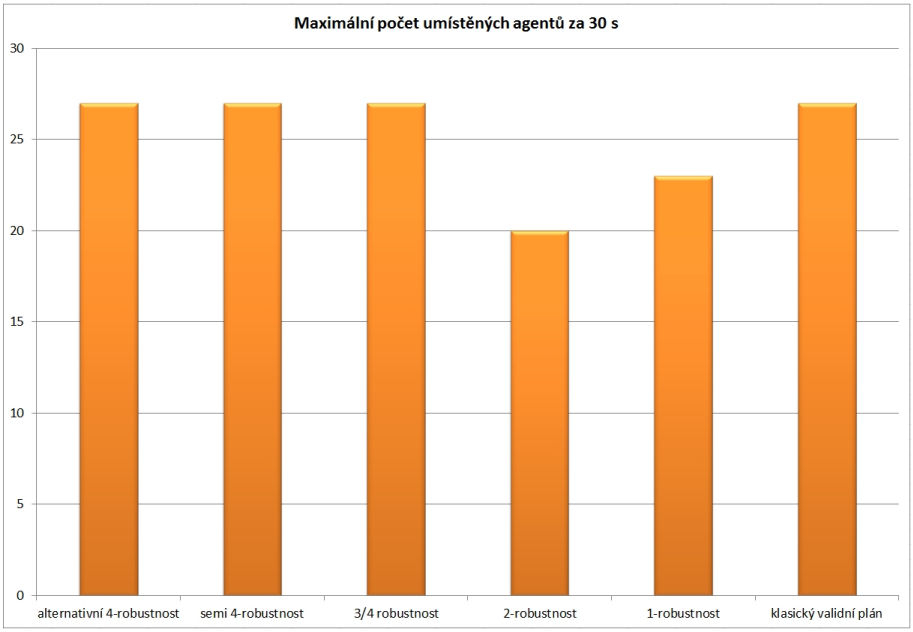

Obrázek 7.12: Medián počtu vyřešených instancí s 10 agenty za 30 sekund – graf DAO mapa.



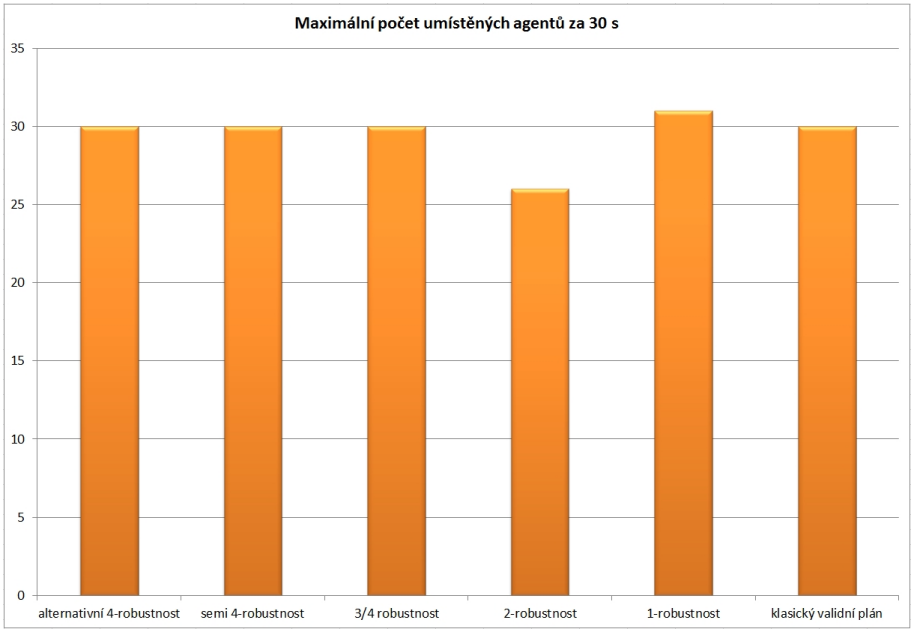

Obrázek 7.13: Medián počtu vyřešených instancí s 10 agenty za 30 sekund – graf Skladiště.




Obrázek 7.14: Medián maximálního počtu umístěných agentů za 30 sekund – graf Mřížka.




Obrázek 7.15: Medián maximálního počtu umístěných agentů za 30 sekund – graf DAO mapa.




Obrázek 7.16: Medián maximálního počtu umístěných agentů za 30 sekund – graf Skladiště.

Ten ale vůbec nebere v potaz zvýšení účelové funkce v důsledku zpoždění! V drtivé většině exekucí 3/4 robustního plánu tak muselo dojít k veškeré eliminaci vzniklého zpoždění a agenti přijížděli do cíle přesně dle svého plánovaného času.

### Rychlost

Zcela rozdílně oproti úspěšnosti vypadají trendy z hlediska rychlosti. Zatímco počty vyřešených instancí u 3/4 robustnosti, alternativní 4-robustnosti a semi 4-robustnosti zůstávají velmi podobné hodnotám klasického validního plánu, v případě 1-robustnosti a 2-robustnosti jsme u grafů Mřížka a DAO mapa svědky značného poklesu, který se prohlubuje s rostoucí hodnotou parametru $k$. V případě Mřížky bylo vyřešeno až 38krát méně instancí. Překvapením je situace v grafu Skladiště, kde nejvíce instancí vyřešil 1-robustní plán. Tento výsledek už nám ale mohl napovědět největší průměrný počet vyřešených instancí, na který jsme upozorňovali dříve.

Co se týče maximálního počtu umístěných agentů, vidíme, že výsledky kopírují trendy z předchozího odstavce, pouze jsou o mnoho vyrovnanější. Jedná se o pěknou ukázku toho, že řešení MAPF problému je těžké a s lineárně se zvětšujícím počtem agentů roste čas potřebný k vyřešení instance exponenciálně.

Z naměřených hodnot dále vyplývá, že není nutné, a dokonce ani žádoucí porovnávat alternativní/semi $k$-robustnost s $k$-robustností pro stejná $k$, protože zatímco v prvním případě lze parametr $k$ zvyšovat pouze za cenu velmi nízkého nárůstu doby běhu i poklesu vyřešených instancí, v případě druhém je tato změna nezanedbatelná a velmi podstatná.

# Závěr

Cílem diplomové práce bylo prozkoumat a navrhnout nové možnosti hledání robustních cest v problému MAPF. V úvodní části práce jsme se seznámili jednak s problematikou MAPF obecně, jednak s doposud známými přístupy k řešení tohoto problému a nastínili jsme jejich výhody a nevýhody. Dále jsme diskutovali možnosti, které se nám nabízí při řešení klasického MAPF problému a představili některé konkrétní algoritmy i jejich případná rozšíření směrem ke zvýšení robustnosti.

V rámci vlastního výzkumu jsme představili několik nových možností, které nám umožní konstruovat plány robustněji. Nejprve jsme se věnovali tvorbě plánů za použití techniky plánovaní s alternativami *(contingency planning)*. Místo klasického sekvenčního plánu jsme navrhli novou podobu plánu s alternativami. Ten se skládá z hlavního plánu, který je na určitých místech doplněn odbočkami neboli alternativními plány a zahrnuje v sobě mnohonásobně více možností průchodu než plán klasický. Tím jsme umožnili jednotlivým agentům, aby si na základě stavu prostředí a velikosti jejich zpoždění zvolili takovou cestu, která co nejvíce eliminuje výskyt kolizí. Zároveň jsme zachovali podobu původního plánu, takže v případě absence zpoždění agenta po cestě zůstane zachována případná optimalita řešení.

Při konstrukci alternativních plánů jsme se zabývali především jejich umístěním, počtem a řešením konfliktů alternativních plánů s hlavními a částečně také konflikty mezi alternativami navzájem. V první metodě, kterou jsme nazvali alternativní $k$-robustnost, dbáme především na to, aby počáteční místa náhradních cest garantovala jejich existenci. Zároveň je vytváříme tak, abychom byli připraveni i na další výskyty zpoždění po vybrání alternativní větve plánu. Kromě základní (volné) varianty jsme vytvořili i striktní variantu řešící do jisté míry kolize mezi alternativami. V druhé metodě nesoucí název semi $k$-robustnost naopak preferujeme odklon od hlavního plánu v co nejzazším termínu i za cenu větší pravděpodobnosti toho, že takový plán nebude existovat. Při konstrukci alternativních cest se zajímáme o bezkolizní průběh při aktuální hodnotě zpoždění.

Navíc jsme ještě hledání alternativních plánů propojili s doplňkovou úpravou plánu pomocí *wait* akcí, která bez zvýšení hodnoty účelové funkce makespan umožní v polynomiálním čase najít a odstranit některá potenciální ohniska konfliktů. Tuto metodu je rovněž možné aplikovat samostatně na jakýkoliv validní plán s relativně dobrými výsledky.

Nakonec jsme se přesunuli ke konstrukci robustních plánů pomocí řízení rychlosti jednotlivých agentů – $min/max$ robustnosti. Úpravou teoretického modelu blíže reálnému světu jsme představili techniku, která za cenu určitého navýšení délky plánu dokáže případné zpoždění během provádění plánu aktivně eliminovat. Vůči pravidelně se opakujícímu zpoždění určité délky pak dokáže garantovat zcela bezkolizní provedení plánu.

Abychom mohli otestovat úspěšnost a použitelnost navržených metod, vytvořili jsme program, který umožňuje simulovat provádění robustních plánů na grafech z běžně používaných MAPF benchmarků. V rámci provedených experimentů jsme ukázali, že $min/max$ robustnost vykazuje velmi vysokou odolnost vůči kolizím a to i v porovnání se všemi doposud známými přístupy. Musíme se

ovšem smířit s určitým nárůstem hodnoty účelové funkce. Robustnost zajištěná alternativními plány naopak zachovává délku plánu na úrovni optima. Co se týče pravděpodobnosti kolize, dá se srovnat s dříve představenými $k$-robustními plány, na druhou stranu je ale hledání alternativních cest podstatně rychlejší než v případě $k$-robustnosti a to i pro dvojnásobné hodnoty parametru $k$ (porovnali jsme 2-robustnost a alternativní/semi 4-robustnost). Dobře funguje i propojení této metody se zvyšováním robustnosti pomocí přidání *wait* akcí. Můžeme tedy říci, že každá z popsaných robustních technik přichází s určitým vylepšením a přispívá tak k podstatnému obohacení této rozvíjející se oblasti MAPF.

# Další možnosti výzkumu

Problematika hledání robustních plánů v MAPF problémech je relativně nová oblast, která zdaleka není vyčerpaná a přináší spoustu možností dalšího pokračování. Co se týče plánů s alternativami, nabízí se jako další rozšíření například konstrukce nového plánu s alternativami, který bude více větvený, takže bude dále obsahovat i odbočky z odboček atd. Rovněž by bylo možné tuto metodu díky její rychlosti využít například při on-line plánování, kde bychom hledali nové cesty nikoliv před provedením plánu, ale přímo během exekuce. Otevřeným problémem nadále zůstává i oblast řešení konfliktů mezi alternativními plány navzájem.

Dále lze problematiku hledání robustních cest přenést i do dalších oblastí mimo klasický MAPF, jako jsou například vážené grafy. V těchto typech grafů by pak mohla být s úspěchem použita metoda $min/max$ robustnosti, kterou jsme v této práci vyzkoušeli na grafech s jednotkovou vahou. Nabízí se i možnost prozkoumání tvorby robustních plánů v prostředí používající jiný model zpožďování. Uvažovat lze například proměnlivé zpoždění jednotlivých agentů nebo konkrétních vrcholů grafu.

# Seznam použité literatury


Multi-Agent Path-Finding (MAPF) Benchmarks. `https://movingai.com/benchmarks/mapf.html`. Navštíveno: 20. 6. 2020.

Pathfinding Benchmarks: File Formats. `https://movingai.com/benchmarks/formats.html`. Navštíveno: 20. 6. 2020.

Picat. `http://picat-lang.org`. Navštíveno: 24. 1. 2020.

Atzmon, D., Stern, R., Felner, A., Wagner, G., Barták, R. a Zhou, N.-F. (2018). Robust Multi-Agent Path Finding. In *Symposium on Combinatorial Search (SoCS)*, pages 2–9.

Atzmon, D., Stern, R., Felner, A., Sturtevant, N. R. a Koenig, S. (2019). Probabilistic Robust Multi-Agent Path Finding. In *SoCS*, pages 162–163.

Atzmon, D., Stern, R., Felner, A., Wagner, G., Barták, R. a Zhou, n.-f. (2020). Robust Multi-Agent Path Finding and Executing. In *Journal of Artificial Intelligence Research*, pages 549–579.

Barták, R., Zhou, n.-f., Stern, R., Boyarski, E. a Surynek, P. (2017). Modeling and Solving the Multi-Agent Pathfinding Problem in Picat. In *IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 959–966.

Dresner, K. a Stone, P. (2008). A Multiagent Approach to Autonomous Intersection Management. *J. Artif. Intell. Res. (JAIR)*, **31**, 591–656.

Hart, P. E., Nilsson, N. J. a Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, **4**(2), 100–107.

Khorshid, M., Holte, R. a Sturtevant, N. (2011). A Polynomial-Time Algorithm for Non-Optimal Multi-Agent Pathfinding. In *SoCS*, pages 76–83.

Kornhauser, D., Miller, G. L. a Spirakis, P. (1984). Coordinating Pebble Motion on Graphs, The Diameter of Permutation Groups, and Applications. In *25th Annual Symposium on Foundations of Computer Science*, pages 241–250. IEEE.

Luna, R. a Bekris, K. (2011). Efficient and Complete Centralized Multi-Robot Path Planning. In *SoCS*, pages 201–202.

Ma, H., Tovey, C., Sharon, G., Kumar, T. a Koenig, S. (2016). Multi-Agent Path Finding with Payload Transfers and the Package-Exchange Robot-Routing Problem. In *AAAI*.

Ma, H., Yang, J., Cohen, L., Kumar, T. a Koenig, S. (2017). Feasibility Study: Moving Non-Homogeneous Teams in Congested Video Game Environments. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 270–272.

Ma, H., Harabor, D., Stuckey, P. J., Li, J. a Koenig, S. (2019). Searching with Consistent Prioritization for Multi-Agent Path Finding. In *AAAI*, pages 7643–7650.

Morris, R., Pasareanu, C., Luckow, K., Malik, W., Ma, H., Kumar, T. a Koenig, S. (2016). Planning, Scheduling and Monitoring for Airport Surface Operations. In *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 608–614.

Pianpak, P., Son, T. C., Toups, Z. O. a Yeoh, W. (2019). A Distributed Solver for Multi-Agent Path Finding Problems. In *Proceedings of the First International Conference on Distributed Artificial Intelligence*, DAIJ '19. Association for Computing Machinery.

Pryor, L. a Collins, G. (1996). Planning for Contingencies: A Decision-Based Approach. *JAIR*, **4**(1), 287–339. ISSN 1076-9757.

Russell, S. a Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*, pages 421–425. Prentice Hall Press, USA, 3rd edition. ISBN 0136042597.

Sharon, G., Stern, R., Goldenberg, M. a Felner, A. (2011). The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 195, pages 662–667.

Sharon, G., Stern, R., Felner, A. a Sturtevant, N. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, **219**, 40–66.

Silver, D. (2005). Cooperative Pathfinding. In *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, page 117–122. AAAI Press.

Stern, R., Sturtevant, N. R., Felner, A., Koenig, S., Ma, H., Walker, T. T., Li, J., Atzmon, D., Cohen, L., Kumar, T. K. S., Boyarski, E. a Bartak, R. (2019). Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *SoCS*, pages 151–158.

Surynek, P. (2010). An Optimization Variant of Multi-Robot Path Planning Is Intractable. In *Proceedings of the National Conference on Artificial Intelligence*.

Wagner, G. a Choset, H. (2011). M*: A complete multirobot path planning algorithm with performance bounds. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3260–3267.

Wagner, G. a Choset, H. (2017). Path Planning for Multiple Agents under Uncertainty. In *ICAPS*.

Wurman, P., D'Andrea, R. a Mountz, M. (2008). Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine*, **29**, 9–20.

Zhou, n.-f., Kjellerstrand, H. a Fruman, J. (2015). *Constraint Solving and Planning with Picat*. Springer International Publishing. ISBN 9783319258812.

# Seznam obrázků

# Seznam tabulek

# Seznam použitých zkratek

MAPF – multi-agent path finding
SoC – sum of costs
WCHA* – Windows Hierarchical A*
TASS – Tree-based agent swapping strategy
CBS – Conflict-Based Search
ITCS – Increasing Cost Tree Search
SAT – Boolean Satisfiability
CSP – Constraint Satisfaction Problem
DAO – Dragon Age: Origins
MAI – minimální alternativní interval

# A. Přílohy

## A.1 MAPFsimulator – uživatelská dokumentace

Program *MAPFsimulator.exe* umožňuje grafickou vizualizaci nalezených řešení MAPF problémů. Dále jej používáme pro spouštění testů, jejichž výsledky uvádíme v kapitole 7.

### A.1.1 Minimální požadavky na systém

Aplikace je vytvořena v programovacím jazyce C#. Pro spuštění a správný běh je vyžadována instalace .NET frameworku ve verzi 4.6 a vyšší. Doporučené rozlišení obrazovky je $1920 \times 1080$ pixelů nebo vyšší.

### A.1.2 Hlavní okno

Po spuštění programu se zobrazí hlavní okno aplikace. To slouží k vytváření jednotlivých MAPF instancí, nastavení parametrů jejich řešení a následné vizualizaci tohoto řešení. Popis všech částí okna je zobrazen na obrázku A.1. Viditelnost některých částí okna se mění v závislosti na stavu, ve kterém se aplikace nachází a zejména po startu programu je většina z nich skryta.

#### Vložení grafu

Nahrání grafu do nové MAPF instance lze provést dvěma způsoby.

1. **Zadáním v okně programu**: v části *Vložení grafu* napíšeme do příslušného řádku graf ve formě textového řetězce a klikneme na tlačítko `Zobrazit graf`. Výsledek můžeme ihned vidět v části okna *Graf*.

   Textový řetězec reprezentující graf zapisujeme na jeden řádek. Pro přístupné vrcholy můžeme použít znaky „**.**“, „**G**“, nebo „**S**“, překážky lze zapsat pomocí znaků „**@**“, „**O**“, „**T**“, nebo „**W**“. Řádky grafu oddělujeme vždy čárkou. Graf musí mít pravidelný (obdélníkový) tvar.

2. **Nahráním ze souboru**: graf vybereme z menu pomocí `Soubor` → `Načíst graf`. Program podporuje soubory s grafy ve formátu, který se používá v rámci MAPF benchmarků (viz Fil). Příklady validních grafů můžeme rovněž nalézt v elektronické příloze této práce.

*Poznámka.* Ačkoliv umožňujeme definovat vrcholy grafu pomocí různých znaků, rozlišujeme v simulátoru jen dva druhy vrcholů – přístupné a nepřístupné (překážky).

#### Vložení agentů

Nahrání startovních a cílových pozic agentů do MAPF instance s grafem lze provést dvěma způsoby.

1. **Zadáním v okně programu**: v části *Vložení agentů* napíšeme do příslušného řádku pozici agenta ve formátu `sX, sY, tX, tY`, kde `sX`, resp. `sY` představuje $x$-ovou, resp. $y$-ovou souřadnici startovního vrcholu a `tX`, resp. `tY` představuje $x$-ovou, resp. $y$-ovou souřadnici cílového vrcholu.

   V souřadném systému číslujeme od nuly a hodnota souřadnic roste zleva doprava a shora dolů. Vrchol $(0, 0)$ se tedy nachází v levém horním rohu. Následně klikneme na tlačítko `Vložit`. Pozice přidaného agenta se ihned zobrazí v seznamu v části *Agenti v grafu*. Zaškrtneme-li zde možnost `Zobrazit agenty v grafu`, objeví se v části *Graf* tyto pozice zvýrazněné barevně.

2. **Nahráním ze souboru**: soubor s pozicemi agentů vybereme z menu pomocí `Soubor` → `Načíst pozice agentů`. Program podporuje soubory ve formátu, které se používají v rámci MAPF benchmarků (viz Ben). Příklady souborů s definicemi pozic agentů nalezneme v elektronické příloze této práce.

   Po vybrání souboru se otevře nové okno, umožňující nám specifikovat počet agentů, který chceme do instance přidat. Vzhled okna vidíme na obrázku A.2. Stiskem tlačítka `Potvrdit` se příslušný počet agentů nahraje do instance. Jejich pozice opět najdeme v části okna *Agenti v grafu*. Zaškrtnutím volby `Vybrat pořadí agentů náhodně` se ze souboru vybere požadovaný počet $n$ agentů náhodně, v opačném případě dojde k nahrání prvních $n$ agentů ze souboru.

V obou případech přidání agentů do programu se u každého z nich kontroluje, zda jeho startovní nebo cílová pozice není v konfliktu s jiným agentem. Pokud tato situace nastane (MAPF instance by tím pádem neměla řešení), zobrazí se okno s příslušným upozorněním a agent do instance nebude vložen.

### Parametry řešení

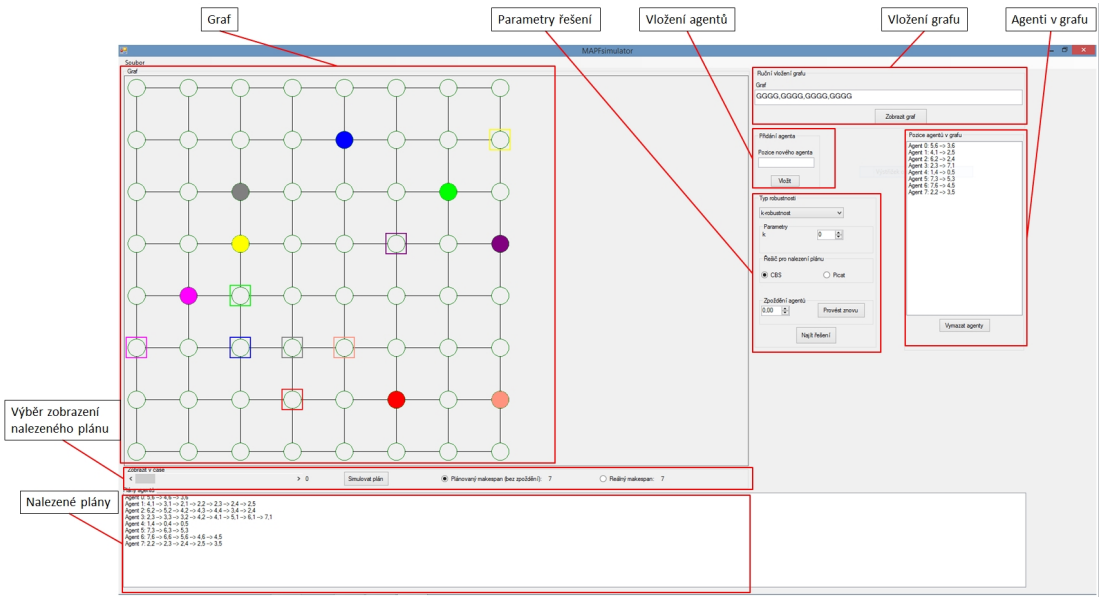
Po vložení grafu a agentů do MAPF instance lze přikročit k nastavení parametrů řešení. Nastavit můžeme:

- `Typ robustnosti` – zde vybíráme jednu z definic robustnosti, kterou bude splňovat nalezené řešení. Na základě výběru konkrétní metody dále nastavíme požadované `Parametry`.

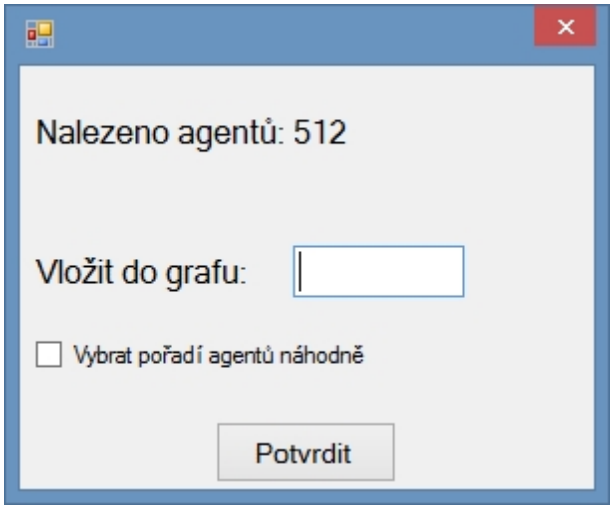
- `Řešič pro nalezení plánu` – řešení můžeme hledat buď pomocí algoritmu CBS, nebo použít externí CSP řešič pro programovací jazyk Picat (stažení řešiče pro různé OS viz Pic). Ve druhém případě je třeba mít správně nastavenou cestu k řešiči (změnu je možné provést v `Soubor` → `Nastavení`). Rovněž je nutné mít v adresáři se spustitelným souborem složku *PicatFiles* s definicemi modelů.

- `Zpoždění` – nastavení hodnoty pravděpodobnosti zpoždění $p_z$ jednotlivých agentů.

Po specifikaci všech hodnot můžeme pokračovat stiskem tlačítka `Najdi řešení`. Program se následně pokusí najít plán se zvolenými vlastnostmi. Časový limit pro hledání lze omezit nastavením příslušné hodnoty v `Soubor` → `Nastavení`. Je-li




Obrázek A.1: Vzhled a popis jednotlivých částí hlavního okna programu.




Obrázek A.2: Vzhled okna s přidáním agentů do grafu.

řešení nalezeno, provede se rovněž exekuce výsledného plánu s daným zpožděním. Po jejím ukončení se zobrazí okno se zprávou o výsledku. Tím je buď úspěšné provedení bez kolizí, nebo neúspěšné provedení s informací o čase a místě výskytu první nalezené kolize.

#### Nalezené plány

Po nalezení řešení MAPF instance se v části *Nalezené plány* zobrazí pro každého z agentů jeho cesta v grafu. Je-li v části *Výběr zobrazení nalezeného plánu* zaškrtnuta volba `Plánovaný makespan (bez zpoždění)`, zobrazí se zde cesty v podobě, jaké je našel plánovač. Zvolíme-li možnost `Reálný makespan`, dostaneme cesty, kterými agenti skutečně projeli v rámci provedení plánu (se započítaným zpožděním a bez dalších alternativních větví).

Cesta každého z agentů je vypsána jako posloupnost vrcholů, které jsou odděleny šipkami. V případě výpisu plánu s alternativami je každá alternativa cesta uvozena znakem „|“, po němž následuje v hranatých závorkách pořadové číslo vrcholu z hlavního plánu, na který se tato cesta napojuje.

#### Výběr zobrazení nalezeného plánu

Jak již bylo napsáno v předchozím odstavci, možnosti `Plánovaný makespan (bez zpoždění)` a `Reálný makespan` ovlivňují podobu výpisu řešení v části *Nalezené plány*. Na posuvníku `Zobraz v čase` poté vybíráme časový úsek, ve kterém chceme graficky znázornit pozice agentů v části *Graf*. Po stisku tlačítka `Simulovat plán` se pomocí animace postupně zobrazí pozice agentů od začátku až do konce.

Rychlost provádění simulace je možné nastavit v `Soubor` → `Nastavení`.

#### Graf

V části *Graf* najdeme nakreslení grafu aktuální MAPF instance a po nalezení řešení zde můžeme také sledovat pozice všech agentů v jednotlivých časových úsecích provedené exekuce. Pozice agenta v grafu je znázorněna pomocí barevného kruhu, jeho cílový vrchol je potom orámován čtvercem stejné barvy.

Stisknutím tlačítka myši v oblasti barevného kruhu zjistíme číselný identifikátor agenta, který se na tomto místě nachází. Identifikátor je shodný s číselným označením agenta v části *Plány* a v části *Agenti v grafu*.

### A.1.3 Okno pro provádění testů

Po výběru položky `Soubor` → `Spustit benchmark` se otevře nové okno, jehož podobu vidíme na obrázku A.3. Před spuštěním konkrétního testu je třeba specifikovat vybrané parametry v levé části okna. S některými z parametrů jsme se již setkali v rámci hlavního okna programu, níže uvádíme význam dalších z nich.

#### Parametry

- `Graf` – po stisknutí tlačítka `Vybrat` se otevře dialogové okno umožňující nahrání souboru s grafem. Po úspěšném načtení se napravo objeví název příslušného souboru.

- `Pozice agentů` – po stisknutí tlačítka `Vybrat` se otevře dialogové okno umožňující nahrání souboru s pozicemi agentů. Po úspěšném načtení se napravo objeví název příslušného souboru s počtem agentů, které program rozpoznal.

- `Počáteční počet agentů` – specifikuje, s kolika agenty v grafu se začne testovat. Minimální počet je 1.

- `Konečný počet agentů` – v průběhu testu je dále možné přidávat do grafu další agenty. Tento parametr určuje horní mez. Hodnota parametru musí být v rozmezí od počátečního počtu agentů do počtu nalezených agentů v rámci scénáře.

- `Krok` – určuje, po kolika agentech se bude přidávat. Začneme-li s $n$ agenty, pak po iteraci se jich v grafu bude nacházet $n +$ `Krok`.

- `Časový limit (ms)` – umožňuje nastavit hranici v milisekundách, po jejímž uplynutí se provádění testů zastaví. Tento parametr nám slouží pro měření rychlosti jednotlivých technik, kdy zkoumáme, kolik instancí dokážeme vyřešit v daném časovém intervalu.

  Ve výchozím nastavení (hodnota 0) není parametr aktivní.

- `Počet opakování se stejným počtem agentů` – kolik se v rámci testu provede instancí se stejným počtem agentů. V každé z instancí budou pozice agentů vybrány náhodně.

  Minimální hodnota je 1.

- `Počet exekucí jednoho plánu` – počet provedených exekucí s nalezeným plánem v dané instanci. V každé z exekucí budou počáteční i koncové pozice agentů stejné, ale změní se místa výskytu zpoždění.

  Minimální hodnota je 1.

- `Počet opakování testu` – kolikrát má být zopakován celý test. Při každém dalším opakování je resetován generátor pro vybírání náhodných instancí na novou hodnotu, takže při $n$ opakováních bude vybráno $n$ náhodných sad instancí.

Spuštění testu zahájíme stiskem tlačítka `Spustit test`. Chceme-li testování ukončit v jeho průběhu, lze tak učinit pomocí tlačítka `Zastavit`. Po jeho stisku dojde k zastavení běhu programu po dokončení právě řešené instance. Průběh testování je možné sledovat v části *Vyřešeno*.

Po skončení testu se naměřené hodnoty zobrazí v části *Výsledky* a automaticky se vloží do schránky ke kopírování. Všechny výsledky v rámci celého běhu programu se pak nacházejí v části *Log*. Tuto část lze skrýt zaškrtnutím položky `Skrýt log`.

### Výsledky

V části *Výsledky* se zobrazují hodnoty makespanu, podílu bezkolizních exekucí a počtu vyřešených instancí. Data uvádíme pro každý test jako průměr přes všechny instance a exekuce v rámci daného počtu agentů.

V případě kliknutí na daný řádek se v části *Instance* objeví výpis všech instancí, které byly v testu pro tento počet agentů provedeny.

### Instance

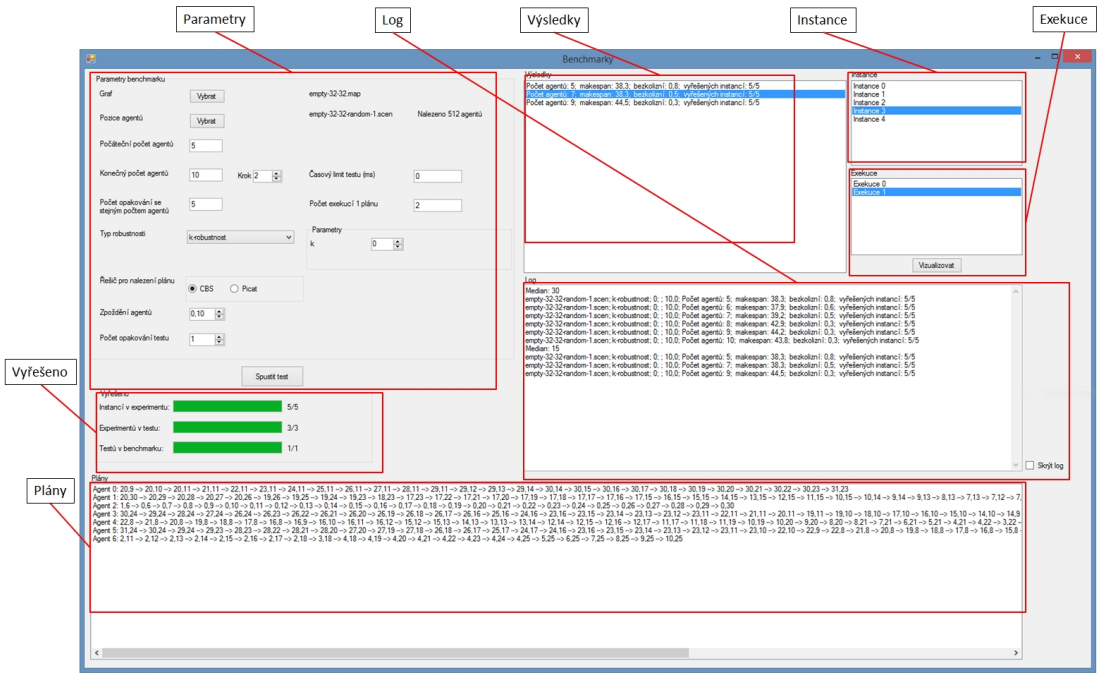
V případě kliknutí na konkrétní položku v seznamu se v části *Exekuce* objeví seznam všech provedených exekucí v rámci instance.

Po poklepání na konkrétní položku v seznamu se v části *Plány* zobrazí výpis cest jednotlivých agentů tak, jak je našel plánovač (tedy bez vlivu zpoždění). Odpovídá volbě `Plánovaný makespan (bez zpoždění)` v hlavním okně programu.

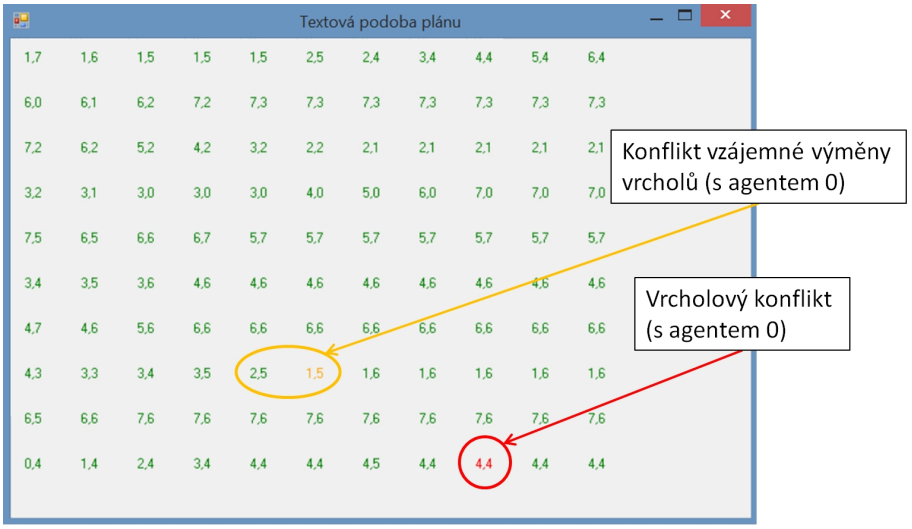
### Exekuce

Po kliknutí na konkrétní položku v seznamu se v části *Exekuce* zpřístupní tlačítko `Vizualizovat`. Po jeho stisknutí dojde k přesunu této instance s příslušnou exekucí do hlavního okna programu s možností její grafické vizualizace.

Po poklepání na konkrétní položku v seznamu se otevře nové okno s posloupností vrcholů, které agenti během této exekuce skutečně projeli. Vzhled okna je uveden na obrázku A.4. Cesta v $i$-tém řádku odpovídá cestě $i$-tého agenta a $j$-tý vrchol v řádku odpovídá pozici v $j$-tém časovém úseku. Vrcholy v tomto okně jsou označeny barevně, zelená barva znamená normální stav, oranžová barva symbolizuje výskyt konfliktu vzájemné výměny vrcholů a červená barva značí vrcholový konflikt agenta v tomto vrcholu.




Obrázek A.3: Vzhled a popis jednotlivých částí okna pro provádění testů.




Obrázek A.4: Vzhled okna s textovou podobou plánu exekuce.

## A.2 Obsah elektronické přílohy

- **MAPFsimulator** – adresář se zdrojovými kódy programu a spustitelným souborem.
- **Dokumentace** – adresář obsahující dokumentaci k programu *MAPFsimulator* vygenerovanou pomocí nástroje *Sandcastle*. Dokumentaci lze prohlížet po otevření souboru `Help/index.html`.
- **Experimenty** – adresář s naměřenými daty, která byla použita v tabulkách a grafech v textu práce.
- **Programátorská dokumentace.pdf** – dokument popisující strukturu programu *MAPFsimulator* a jeho celkové fungování.