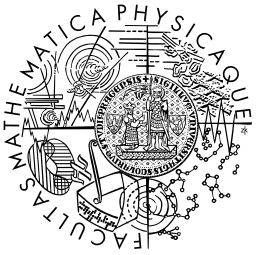


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

**BAKALÁŘSKÁ PRÁCE**

Ivan Krasičenko

# Multiagentní hledání cest na Ozobotech

Katedra teoretické informatiky a matematické logiky

| | |
|---|---|
| Vedoucí bakalářské práce: | prof. RNDr. Roman Barták, Ph.D. |
| Studijní program: | Informatika |
| Studijní obor: | Programování a softwarové systémy |

Praha 2018

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

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

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

Chtěl bych poděkovat rodičům, že mě vždy podporovali. Bohu, že mě v tom nenechavá samotného. Učitelem ze školy za jejich trpělivost a přívětivost. Všem spolužákům a spolubydlícím se kterými jsem měl tu čest trávit čas.

Název práce: Multiagentní hledání cest na Ozobotech

Autor: Ivan Krasičenko

Katedra: Katedra teoretické informatiky a matematické logiky

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

Abstrakt: Vytvořeni testovacího simulačního prostředí pro multiagentní hledání cest na Ozobotech. Program určen pro použiti na mřížkových mapách. Vytvářeni a ukládaní instanci problému. Použiti programovacího jazyka Picat pro nalezení cesty pro Ozoboty. Naprogramováni a testování několika řešících technik. Použiti no-swap, 1-robustnost. Zahrnuta varianta, kde k pozici se modeluje i směr natočeni robota. Porovnávaní makespanu simulovaného běhu a běhu spuštěného na Ozobotech.

Klíčová slova: Ozoboty Multi-agentní hledání cest Mřížkové mapy Simulace

Title: Multi-agent Path Finding on Ozobots

Author: Ivan Krasičenko

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: Creating of test simulation environment for multi-agent path finding on Ozobots. Program is designed for grid maps. Can load and strore instantion of problem. Program language Picat is used to find path for Ozobots. Some MAPF solvers are programmed and used. Use of no-swap, 1-robustness. Includes variant, where position with robot rotation is modeled. Compares makespan of simulaton and real run on Ozobots.

Keywords: Ozobots MAPF problems Grid maps Simulation

# Obsah

# 1. Úvod

Znát pravdu je velice důležité. Každý den se setkáváme s řadou problémů, které potřebujeme pochopit, namodelovat, vyřešit, a řešení aplikovat na problém. Pokud náš model je špatně navržen, nebo pokud problém špatně namodelujeme, získané řešení nemusí mít očekávaný dopad. V této práci se budeme zabývat hledáním cest pro více agentů na mřížkové mapě. Řešení získané na základě této práce lze aplikovat v každodenním životě, konkrétně v průmyslových halách. Pro uvedení příkladu, představme si halu, ve které máme sadu palet. Taktéž máme sadu robotů, které dovedou s paletami pohybovat. Robot vždy dokáže zajet pod paletu, zvednout ji a přemístit na jiné místo, kde ji opět položí. Kdybychom měli sklad s takovými paletami a jednoho takového robota, můžeme pro něho vybrat startovní a cílovou pozici, vzít v potaz všechny překážky v skladu a najít nejrychlejší cestu ze startu do cíle. Když budeme postup opakovat a jednou za čas robotovi řekneme, aby paletu zvedl nebo položil, tak získáme způsob, jak efektivně řídit sklad bez zbytečných prostojů. Pokud bychom chtěli běh skladu více zrychlit, například ve vytížených ročních obdobích, jako jsou Vánoce, může si společnost pořídit několik robotů navíc robotu. Ale když pustíme všechny roboty ve stejný čas, najednou pro nějakého robota vznikne určité množství pohybujících se překážek, kterým se musí vyhnout. Což může být vyřešeno zavedením techniky se senzory, kde si roboti budou hlídat, zda cesta před nimi je průchozí. Toto řešeni ovšem není dokonalé, a snadno vymyslíme rozmístění robotů, které se zvolenou technikou selže. Proto jsme si zavedli centrální plánovaní, ve kterém jsme si na začátku plánu určili startovní pozice pro roboty, vybrali použitelné cesty a cílové pozice pro roboty. Pak už jen následuje výpočet. Ukázalo se, že tento problém je těžký. Proto se hledají modely, jak sklad namodelovat, což je možné modelovat mnoha různými způsoby. Pokud je model obsáhlejší, zabere více času na výpočet. Jednodušší modely na druhou stranu nemusí popisovat skutečnost natolik dobře, aby se daný model dal v realitě vůbec použít. Obtíže pro určení kvalitního modelu přináší i samotná realita. Vytvořený model na konkrétní problém nám nemusí nic říkat o tom, model bude reagovat v reálném světě. Zjišťování, jak moc je model v reálem světě vhodný, nejde jinak než prováděním experimentů a testováním toho, nakolik se skutečné výsledky liší od těch očekávaných. Testování stoji čas a úsilí. Je potřeba obstarat roboty, prostor, programování, testovací vybavení. A každý samotný test spotřebuje nějaký čas a energii robota. Testování modelu je i finančně náročné.

V této práci se budeme zabývat miniaturními roboty Ozobot Evo. Budeme testovat různé modely v různých reálných podmínkách. Pro tento účel vzniklo softwarové dílo MAPF Scenario, které uceleně řeší vytváření mřížkových map, umisťování robotů na pozice, spouštění řešících technik a interaktivní zobrazování výsledků s možností simulace. Také umožňuje generovat soubor s instrukcemi pro ozobota, zobrazovat, ukládat a tisknout mapu. K tomu bylo přidáno rozhraní pro testování ozobota přímo na displeji monitoru. Tímto MAPF Scenario se snaží usnadnit testování a ladění různých řešících technik. Představíme 4 řešící techniky a 2 konfigurace robotu. Porovnáme dobu běhu reálných robotů s dobou simulovanou.

Výstupem práce by měl být model, jehož aplikace v reálném světě by mohla

usnadnit provoz robotů manipulujících s paletami ve skladech. Použití vhodného modelu by mohlo zvýšit efektivitu pohybu robotů ve skladech, snížit náklady na lidskou pracovní sílu, snížit prostoje, a rychlejší dopravou palet by mohl pomoct společnostem k rychlejšímu zpracování nebo doručení zakázek a následnému zvýšení jejich obratu.

# 2. Formalizace MAPF problému

## 2.1 Formální popis problému

Mějme plochu, na které je vyznačena pravidelná čtvercová mřížka. Průsečíky čar na mřížce budeme nazývat vrcholy a úsečky mezi nejbližšími vrcholy budeme nazývat hrany. Některé vrcholy nebo hrany mohou být označené jako neprůchozí. Všechny hrany na ploše mají stejnou délku. Takto popsanou plochu budeme nazývat mřížkovou plochou.

Mějme množinu robotu, které jsou umístěny na mřížkové ploše, a chceme aby se dostaly do cílových vrcholů. Roboti máji společnou množinu akcí, které mohou provádět. Výsledkem akce pro nějakého robota může, ale nemusí, být změna vrcholu, ve kterém se nachází. Plán $P_i$ pro robota $i$ definujeme jako posloupnost akcí, které když robot provede, tak se dostane ze startovního vrcholu do cílového vrcholu. Akce pro robota je posloupnost instrukcí, která se nějak odráží v čase nebo pozici robota. Nejjednodušší množinou akcí může být { vpřed, vlevo, vpravo, čekej }. Každá akce má předem známou dobu trvání. *Během* budeme rozumět časové rozmezí od okamžiku, kdy roboti na svých startovních pozicích začaly vykonávat akce, až do okamžiku, kdy všichni roboti přestaly akce vykonávat a jsou v cílových vrcholech. Množina akcí pro roboty není jednoznačně určena a její podobu lze vhodně volit. Zpravidla se množina akcí volí tak, aby umožnila vytvořit úspěšný plán.

Formálně mřížkovou plochu můžeme popsat pomocí grafu $G = (V,E,d)$, kde $V$ je množina vrcholu mřížkové mapy, $E$ je množina hran mřížkové mapy a $d$ je délka hrany. Roboty popíšeme jako uspořádanou trojici R = ($Rs$,$Rc$,$Act$), kde $Rs$, $Rc$ jsou uspořádané n-tice, kde, $Rs_i \in V$ určuje vrchol, ve kterém robot $i$ startuje, $Rc_i \in V$ určuje vrchol, ve kterém chceme, aby se robot nacházelpo provedení všech akci. A $Act$ je seznam akcí, které je schopen každý robot ze skupiny vykonávat.

## 2.2 MAPF řešiče

Definujeme si *MAPF řešič* jako algoritmus, který na vstupu bere graf $G$, a uspořádanou n-tici agentu. Každý agent má svůj startovní a cílový vrchol. Výstupem algoritmu je seznam cest $C$, kde pro agenta $i$ cesta $C_i$ je cesta v grafu $G$ vedoucí ze startovního vrcholu do cílového. Navíc musí splňovat podmínku, že více agentů ve stejném čase nesmí sdílet stejný vrchol. *Vnořeným řešičem* budeme rozumět algoritmus, který běží na pozadí picatu (Barták a kol., 2017) , prohledává stavový prostor a snaží se najít řešení. Přesněji tím budeme myslet algoritmus pro řešení SAT, MIP nebo CSP.

Vstupem MAPF řešiče budeme nazývat *MAPF problém* a tvoří ho orientovaný graf $G = (V,E)$ a uspořádaná n-tice agentů $a_1, ..., a_k$. Každému agentovi $a_i$ je přiřazen počáteční vrchol $s_i \in V$ a cílový vrchol $c_i \in V$. Výstupem řešiče je pro každého agenta $i$ cesta $C_i = (s_i = C_i[0], C_i[1], ..., C_i[m], C_i[m+1] = c_i)$. Budeme značit $C_i[t]$ jako vrchol na cestě agenta $i$ v čase $t$. $m+1$ označíme délku cesty. Agent se při změně času může posunout z vrcholu do souseda, a nebo pozici neměnit. Formálně, pokud $u = C_i[t]$ a $v = C_i[t+1]$ pak nutně $(u,v) \in E$. Setrvání

agenta ve vrcholu $v$ i v následujícím čase se dá řešit přidáním reflexivních hran do grafu $G$. Neboli $\forall v \in V$ platí, že $(v,v) \in E$. Taktéž musí platit, že dva různí agenti nesmí ve stejný čas být v jednom vrcholu, neboli $\forall i \neq j, \forall t$ platí $C_i[t] \neq C_j[t]$.

Samotné implementace MAPF řešičů se dělí na dvě základní kategorie:

1. **Řešiče založené na redukci.** Tato skupina řeší MAPF problém tak, že ho převede na jiný problém, třeba na SAT (Surynek, 2012) nebo na celočíselné lineární programování (Yu a LaValle, 2013), a nebo answer set programming (ASP) (Surynek, 2012).

2. **Řešiče založeny na prohledávaní.** Ty jsou založeny na prohledávaní v grafu. Zahrnuji varianty algoritmu A*. Jiné přístupy využívají prohledávací stromy (Surynek a kol., 2017)

Zde se budeme zabývat řešičem založeném na redukci, a to konkrétně tím, který používá převod na SAT.

## 2.3 MAPF řešič v Picatu

Picat je víceúčelový programovací jazyk, který poskytuje několik nástrojů pro modelování a řešení kombinatorických problémů (Zhou a kol., 2015). Tím že MAPF problém popíšeme pomocí Picatu, získáme prakticky zadarmo převod do SAT, CSP či MIP. A tím možnost testovat různé řešiče.

Picat vychází z Prologu. Podobně jako prolog má pravidla, která mají hlavu a tělo. Hlava je splněna pokud jsou splněny všechny predikáty v těle. Na rozdíl od Prologu, Picat má k dispozici syntaxi pro zápis problémů s omezujícími podmínkami a tři různé řešiče takových problémů. Zde se budeme zabývat formulací MAPF problému v Picatu, a samotné překlady do instancí problému ponecháme na Picatu.

MAPF řešiče se dále dají dělit podle vlastnosti řešení, které naleznou.

### 2.3.1 Děleni MAPF řešiču

Definujeme si akční délku cesty. Nechť MAPF řešič pro daný graf $G$ a seznam agentů $As$ dá cestu $C_i = (s_i = C_i[0], ..., C_i[t] = c_i)$, kde $s_i$ je počáteční vrchol a $c_i$ cílový vrchol agenta $i$. Pak jako *akční cestu* budeme brát cestu z $s_i$ do vrcholu $C_i[x]$ takovou, že $C_i[x] = c_i$ a současně $C_i[x-1] \neq c_i$ a navíc $\forall j \in \{x,..,t\}$ pláti, že $C_i[j] = c_i$. Dalo by se říci, že akční cesta je cesta ze startovního vrcholu do prvního navštívení cíle. A současně pak agent do konce cesty v cíli zůstane. *Délku akční cesty* pak budeme brát jako délku akční cesty, a budeme ji značit $|C_i|_d$.

Jednou z vlastností, kterou chceme aby řešení mělo, je co nejkratší doba, kterou agenti potřebují, aby se dostaly ze svých startovních vrcholů do cílových. Pro seznam cest, které MAPF řešič nalezne, zavedeme *Dobu běhu* jako tu nejdelší ze všech nalezených cest. Formálně doba běhu je $\max_{i \in As} |C_i|_d$. Při optimalizaci doby běhu hledáme řešení s nejnižší dobou běhu. Tato řešení nás mohou zajímat, pokud optimalizujeme celkový čas a nejsou pro nás tolik důležité jednotlivé zdroje, které agenti při pohybu spotřebují.

Další optimalizovanou vlastností může byt *Součet cen*, ten se snaží najít nejmenší celkový počet navštívených vrcholů. Tedy součet cen bude $\sum_{i \in As} |C_i|_d$.

Dané kritérium hledá nejnižší součet cen. V tomto případě se snažíme, aby agenti udělaly co nejmenší počet kroků. A to i za cenu, že celková dobu běhu může být vyšší.

Můžeme také optimalizovat podle *priorit*. Zde bereme v potaz, že pohyb některých agentů je dražší než jiných. Třeba agent $A$ může mít lehký náklad a nevadí, když se více projede po mapě, zatímco agent $B$ může mít těžký náklad a je žádoucí, aby náklad dovezl do cíle v co nejmenším počtu kroků.

Také lze hledat cesty v grafu, které mají jistou robustnost. *K-robustností* se rozumí to, že předtím, než agent $A$ přejde do vrcholu $V$, tak v daném vrcholu nikdo nesměl posledních k kroků být. Tato varianta umožňuje nadále hledat optimální dobu běhu i součet cen, jen vynucuje další vlastnost nalezených cest. Tento typ řešení se může hodit pro agenty, kteří z nějakých důvodu nemohou plnit požadavky na přechod mezi vrcholy v čas. K robustnost umožňuje najít cesty, které by byly vhodné pro tento typ agentů.

Dále můžeme požadovat, aby agenti při průchodu mezi vrcholy nesdílely hrany. Tomuto omezení se říká *omezení hranových konfliktů*. Hodí se obzvlášť pokud agenty jsou roboti, a cesty jsou úzké uličky, kde se roboti nemohou navzájem vyhnout. Formálně, pokud agent $i$ je ve vrcholu $C_i[T]$ v čase $T$, a v čase $T+1$ je ve vrcholu $C_i[T+1]$, tak $\forall j \in As, j \neq i$ platí $C_j[T] \neq C_i[T+1]$ nebo $C_j[T+1] \neq C_i[T]$.

*Zamezení vláčkového pohybu.* Toto omezení je nejvíce patrné v úloze Loydova patnáctka. Máme 16 poliček, 15 kamenů a 1 volnou pozici. Můžeme vždy pohnout jedním kamenem, a to jedině na volnou pozici. Toto omezeni vede na 1-robustnost.

### 2.3.2 MAPF řešič s optimální dobou běhu

Uvedeme si Picati program, který bude řešit MAPF problém a následně si ho vysvětlíme.

```
%Graph (vstup) je graf reprezentovaný seznamem sousedů [(v,sousede)],
%  kde v je vrchol grafu, a sousede je seznam jeho sousedu
%As (vstup) je seznam dvojic informací o agentu [(as,ac)],
%  kde as je startovní pozice agenta, ac je cílová pozice agenta
%B (vystup) je tridimenzionální pole B[T,A,V],
%  které nabývá hodnotu 1 pokud agent A v čase T je ve vrcholu V.
%  Jinak nabývá 0.
path(Graph,As,B) =>
        N = len(Graph), % počet vrcholů v grafu
        K = len(As), % počet agentů
        lower_upper_bounds(Graph,As,LB,UB),
        between(LB,UB,M),
        B = new_array(M+1,K,N),
        B :: 0 .. 1,

        %(1) zavedení počáteční a koncové podmínky
        foreach(A in 1..K)
                (V,FV) = As[A],
                B[1,A,V] = 1, % agent začíná na své startovní pozici.
                B[M+1,A,FV] = 1 % agent končí ve své koncové pozici
        end,
```

```
%(2) agent A v case T se nachází právě v jednom vrcholu V
foreach(T in 1..M+1, A in 1..K)
        sum([B[T,A,V]: V in 1..N]) #= 1
end,

%(3) dva a více agentů nesdílí společný vrchol
foreach(T in 1..M+1, V in 1..N)
        sum([B[T,A,V]: A in 1..K]) #=< 1
end,

%(4) agenty mohou změnit svůj vrchol na jiný jen pokud je tam hrana.
foreach(T in 1..M,A in 1..K, V in 1..N)
        Neibs = Graph[V],
        B[T,A,V] #=>
        sum([B[T+1,A,U] : U in Neibs]) #>= 1
end,
solve(B).
```

Myšlenka je následující. Vytvoříme si trojdimenzionální pole $B[T,A,V]$ s množinou možných hodnot $\{0,1\}$. První dimenzi označíme jako čas $T$, druhou jako číslo agenta $A$, a třetí jako číslo vrcholu $V$. Pokud $B[T,A,V] = 1$, tak to znamená, že v čase $T$ je agent $A$ ve vrcholu $V$. Pokud $B[T,A,V] = 0$, tak agent ve vrcholu není. Pro lepší představu, pokud bychom měli jen jednoho agenta, tak dostaneme dvou dimenzionální pole $B[T,1,V]$. A procházením tohoto pole podle času můžeme zjistit v jakých vrcholech se agent zrovna nachází. Vytvoření takového pole je provedeno následujícími řádky.

```
1. B = new_array(M+1,K,N),
2. B :: 0 .. 1,
```

První řádek vytvoří pole potřebné velikosti a druhý vymezí doménu pro proměnné, které se využijí při hledání omezujících podmínek.

Úsek kódu, označený jako (1), omezuje počáteční a cílové podmínky pro agenty. Přesně říká, že každý agent se v čase 1 nachází v počátečním vrcholu. A v čase $M + 1$ v cílovém vrcholu. Foreach by se zde dal spíše brát jako zkrácená varianta místo výčtu podmínek pro každého agenta. V úseku kódu (2), by se na podmínku dalo nahlížet jako: „V jednom konkrétním čase jeden konkrétní agent smí být právě v jednom vrcholu“. Tato podmínka vyloučí takové situace, kdy by agent najednou v čase $T$ přestal existovat, nebo kdyby najednou byl ve více vrcholech současně. Vymezení (3) říká, že agenti nesměji sdílet vrcholy. Doslova, že když spočtu počet agentů v čase $T$, ve vrcholu $V$, tak jejich počet je ze shora omezen jedničkou. Čtvrtý úsek (4) bere v potaz vstupní graf. Říká, že pokud je agent $A$ v čase $T$ je ve vrcholu $V$, tak v následujícím čase musí být v jednom ze svých sousedů. Zde bereme v potaz, že $V$ je svým vlastním sousedem. Zde je dobré si poznamenat rozdíl mezi operátory >, <,>=, =<, = a operátory # >, # <,# >=,# <=,# =, neboli operátory s a bez #. Ty bez # určují běh picatího programu, testují levou a pravou stranu výrazu a případně uspějí nebo neuspějí. Zatímco operátory s # určují podobu omezujících podmínek, které bude používat vnořený řešič. V budoucnu se také setkáme s operátorem # =>, což je operátor pro klasickou implikaci určený pro definování omezující

podmínky. Příkaz *solve*($B$) spustí příslušný vnořený řešič a jeho výsledkem bude pole $B$, které splňuje omezující podmínky, a nebo false, pokud pole, které splňuje všechny podmínky, neexistuje. Přesné chování programu a popis příkazu lze najít v dokumentaci Picatu.

Dosud jsme se soustředili pouze na popis řešení přípustného. Teď z něj uděláme řešení optimální. V tomto řešiči budeme optimalizovat dobu běhu. O to se starají následující řádky.

```
        lower_upper_bounds(Graph,As,LB,UB),
        between(LB,UB,M),
```

Je za nimi následující myšlenka. Funkce *lower_upper_bounds*($Gr$,$As$,$LB$,$UB$) odhadne nejkratší a nejdelší možné řešení problému. $LB$ je zkratka za „Lower Bound“, a značí dolní mez. Kdežto $UV$ - „Upper Bound “ je horní mez. Potom picati kód bude postupně hledat přípustná řešení od nejkratšího možného času až po nejdelší možný. S tím, že pokud pro čas $M$ nebude existovat přípustné řešení, tak se $M$ zvýší o jedna a zkusí znovu najít přípustné řešení. A pokud řešení nalezne, tak ho vrátí. Tím pádem vrátí nejkratší možné řešení, neboť kdyby existovalo kratší, tak by ho picat našel v minulém kroku při hledání kratšího řešení. Funkce *between*($LB$,$UB$,$M$) se chová tak, že nejprve za počáteční $M$ zvolí hodnotu $LB$. Následně pikat zkouší splnit predikáty napsané po predikátu *beetween*, a pokud ty selžou (mezi nimi je i predikát *solve*($B$)), tak funkce dosadí do $M$ hodnotu o 1 větší. Takto pokračuje až do hodnoty $UB$, a pokud neuspěje ani ta, tak celý predikát vrátí false. Jelikož řešení neexistuje.

### 2.3.3 MAPF řešič s optimalizovaným součtem cen

Nyní popíšeme modifikaci výše uvedeného řešiče, který nám bude hledat nejnižší součet cen. K tomu zavedeme pomocný predikát *end_time*($B$,$As$,$K$,$M$,$E$), který bude vypadat následovně:

```
%As (vstup) je seznam dvojic informací o agentu [(as,ac)],
%  kde as je startovní pozice agenta, ac je cílová pozice agenta
%B (vstup) je třidimenzionální pole B[T,A,V],
%  které nabývá hodnotu 1 pokud agent A v čase T je ve vrcholu V.
%  Jinak nabývá 0.
%K (vstup) Je počet agentů
%M (vstup) je délka plánu (neboli maximální čas T)
%E (vystup) je pole indexované agenty, kde E[A] udává akční délku cesty,
%  jak jsme ji definovali vyše,  pro agenta A.
end_time(B,As,K,M,E) =>
        E = new_array(K),
        E :: 1..M+1,
        foreach (A in 1..K, T in 1..M+1)
                (V,FV) = As[A],
        % (5) pokud agent dosáhl délku akční cesty v čase T,
        % tak v předchozím čase ještě v cíli nebyl.
                if T > 1 then
                        E[A] #= T #=> B[T-1,A,FV] #= 0
                end,
        % (6) pro všchny časy po dosažení délky akční cesty
```

```
        % platí, že agent zůstává v cílovém vrcholu
        foreach (T1 in T..M+1)
                        E[A] #= T #=> B[T1,A,FV] #= 1
                end
    end.
```

Myšlenka tohoto predikátu spočívá v tom, že nám vymezí pole s délkami akčních cest jednotlivých agentů. Formálně, pokud řešič vrátí pro agenta $i$ cestu $C_i$ a délka akční cesty bude $d_i = |C_i|_d$, pak $E[I] = d_i$. Úsek (5) zajišťuje, že koncová doba $E[i]$ je nejmenší možná. Neboli $C_i[E[i] - 1] \neq c_i$. A úsek (6) hlídá, že po doražení do cíle agent v cíli zůstane. Neboli $\forall T \in \{E[I]..M + 1\}$ platí, že $C_i[T] = c_i$. Pak můžeme volat rozšířenou funkci predikátu *solve*, konkretně $solve([\$min(sum(E))],B)$. První argument funkce *solve* bere seznam voleb pro vnořený řešič. V tomto konkrétním případě mu říká, že chce takové řešení, kde součet prvků z pole $E$ je minimální. V této variantě funkci *between* nebudeme používat, a jako $M$ si rovnou dosadíme $UB$. Toto řešení je poněkud naivní, neboť počítá rovnou s velkým množstvím proměnných. Lepší řešení bylo navrhnuto v (Barták a kol., 2017), kde se doba běhu zvyšuje postupně spolu se součtem cen. Poznamenáme, že symbol \$ v zápisu $solve([\$min(sum(E))],B)$ značí že *min* je term, a nikoliv volání funkce.

### 2.3.4 MAPF řešič optimalizující priority

MAPF řešič, který počítá součet cen, předpokládá, že každý přesun agenta po hraně stojí stejnou cenu. Můžeme zavést vektor vah $w = (w_1,w_2,...w_k)$, kde $w_i$ bude cena přechodu po hraně pro agenta $i$. Řešič optimalizující priority bude volat jinou optimalizační funkci, a to tu, která akční délku cesty pro agenta $i$ přinásobí $w_i$. Tedy $min(sum(WE))$, kde $WE$ je vektor, kde $(WE)_i = w_i * E[i]$.

### 2.3.5 L-robustní MAPF řešič

Tento řešič vychází z úpravy předchozích řešičů, a to úpravou podmínky (3) následujícím způsobem:

```
%pokud je agent A v case T na pozici V,
%  tak zadny jiny agent nebyl v case T-L az T na pozici V
foreach(T in 1..M1, A in 1..K, V in 1..N)
        B[T,A,V] #=> sum([B[Prev,A2,V] : A2 in 1..K, A2!=A,
                                Prev in max(1,T-L)..T]) #= 0
end
```

Tento kód by se dál přečíst takto: pokud agent $A$ je v čase $T$ ve vrcholu $V$, tak v čase $T - L$ až $T$ tam nesměl byt žádný jiný agent, kromě agenta $A$. Pro upřesnění, využívá se zde syntaxe podobná hasklu, kde kód $[x : x \text{ in } 1..N, podminky]$ se podobá klasickému matematickému definování množiny vymezením jejich vlastností.

### 2.3.6 MAPF řešič bez hranových konfliktů

Zamezení hranových konfliktů lze docílit z předchozích řešičů přidáním následujícího úseku kódu:

```
 %dva různí agenti nesmí přejít přes společnou hranu
foreach(T in 1..M, V in 1..N, A in 1..K)
        foreach(U in Graph[V],C in 1..K)
                if A != C then
                        (B[T,A,V] #/\ B[T+1,A,U])  #=>
                                B[T,C,U] + B[T+1,C,V] #=< 1
                end
        end
end,
```

Tento úsek kódu pro každého agenta $i$, který přechází z vrcholu $C_i[T]$ do vrcholu $C_i[T+1]$ přidá podmínku, že agent $j \neq i$ nesmí byt ve vrcholu $C_i[T+1]$ v čase $T$, a v čase $T+1$ nesmí být ve vrcholu $C_i[T]$). Tedy $C_i[T+1] \neq C_j[T]$ a současně $C_i[T] \neq C_j[T+1]$. Tento úsek kódu vede na velké množství podmínek.

### 2.3.7 MAPF řešič dle doby běhu a součtu cen (DB+SC)

Vezměme řešič pro optimální dobu běhu. Ten postupně hledá řešení od nejkratší možné doby běhu až po nejdelší doby běhu. Můžeme při hledání řešení délky $t$ zahrnout požadavek na minimalizaci počtu cest. Výsledek takového řešiče bude nadále optimalizovat dobu běhu, ale současně agenti, jejichž akční délka cesty je kratší než doba běhu, nebudou provádět zbytečné kroky.

Upravený algoritmus by mohl vypadat následujícím způsobem:

```
path(Graph,As,B)
        N = len(Graph),
    K = len(As), % pocet agentu
        lower_upper_bounds(Graph,As,LB,UB),
        between(LB,UB,M),
        B = new_array(M+1,K,N), %B = (Cas,Agenty,Vrcholy)
        B :: 0 .. 1,

        end_time(B,As,K,M,E),
        .
        .
        .
        solve([$min(sum(E))],B). /* $min znaci term */
```

Program nejprve zkouší najít řešení od nejmenší doby běhu $LB$, až po nejvyšší dobu běhu $UB$, a pro danou dobu $M$ se snaží pro každého agenta minimalizovat jeho délku akční cesty.

## 2.4 Transformace Svět na MAPF a MAPF na Akce

Je potřeba zavést transformaci z mřížkové mapy s roboty do vstupu pro MAPF řešič, tedy do orientovaného grafu a n-tice agentu. Tuto transformaci pojmenujeme jako transformace Svět→MAPF nebo zkráceně SM. Formálně Svět se bude bude skládat z mřížkové plochy $G_s$ a uspořádané trojice $R = (Rs,Rc,Act)$. MAPF problém bude Graf $G_m$ a seznam agentů $A$. Transformace Svět→MAPF bude dvojice $(Gf,Vf)$, kde $Gf$ je algoritmus, který vezme graf $G_s$ a vytvoří z něj graf $G_m$. A $Vf$ je zobrazení z množiny vrcholu $G_s$ do množiny vrcholu $G_m$. Formálně zapsáno $G_m = Gf(G_s)$ a $Vf : V(G_s) \to V(G_m)$.

Následně zavedeme transformaci z výstupu MAPF řešiče do akci jednotlivých robotů, a označíme ji jako MAPF→Akce zkráceně MA. Výstupem MAPF řešiče je seznam cest v grafu $G_m$. Pro agenta $i$ dá MAPF řešič cestu $C_i$. Výstupem transformace je seznam posloupnosti akcí pro nějakou množinu robotů $R = (Rs,Rc,Act)$. Neboli pro robota $i$ nám vrátí posloupnost $A_i = \{a_t\}_1^{tkonec}$, kde $a_t \in Act$. Samotná transformace pak bude zobrazení k-te r-tice vrcholu z cesty do nějaké akce. Formálně $MA : v^r \to a$, kde $v \in V(G_m), a \in Act$. Zavedeme si pomocnou funkci $usek(C_i,k,r)$, která pro cestu $C_i = (C_i[0],...,C_i[tkonec])$ vrátí úsek cesty $(C_i[k],...C_i[k+r])$, $r > 0$, $r \in \mathbb{N}$. Pak pro agenta $i$ a jeho cestu $C_i$ dostanu k-tou akci pomocí úseku $u_k = usek(C_i,k,r), r > 0$, který zobrazím za pomocí $MA$ do množiny akcí $Act$.

Konkretní metoda pro vytvoření plánu pro mřížkovou plochu $M$ a pro roboty $R = (Rs,Rc,Act)$ se bude skládat z transformace Svět→MAPF, konkretního MAPF řešiče, a transformace MAPF→Akce.

# 3. Připrava prostředi

Naplní vlastní práce spočíva ve vybětu několika reprezentci světa, několika metod a otestování, jak si metody vedou v různých světech.

## 3.1 Průběh testování v MAPF Scenario

Provedeni testu se dá rozdělit na následujici sadu kroku.

1. **Navrhnout svět.** MAPF scenario umožňuje nadefinovat plošku pro roboty a zvolit jejích výchozí pozice.

2. **Zafixovat délku hrány a délky operaci.** Vysledký testu jsou závisle na délce hrány v plošce, a době trvání jednotlivých akci robotu. Délka hrány se nastavuje v programu scenaro, a délka akci se nastavuje ručně ve vygenrováném kodu pro ozobota. Současné nastavení nedovoluje generovát mřižku větší než 30 na 30 vrcholu.

3. **Zvolit řešici metodu a vygenerovat řešeni.** Samotné řešici metody jsou napsané v jazyce Picat. Jejich seznám je schrnut v souboru picat_iface. Scenario umožňuje některou z těchto metod zvolit a spustit. Před zavoláním řešene metody scenario vytvoři soubor s grafem problému a s pozici agentu. Po zavolání metody se cenario se pokusi přečist soubor s řešením, který metoda vygenerovála.

4. **Vygenerovat programy pro ozoboty.** Metodou vygenerováné řešení obsahuje posloupnosti instrukci pro jednotlivé roboty. Generování programu pro ozoboty se provádi metodou vkládaní časti kodu do existujici šablony. Šablona je pojmenována „template.ozocode“ a musi obsahovat funkce „ENTRYPOINT“, „INJECTED“ a taktéž volání funkce „ENTRYPOINT“. Samotné vkládaní kodu spočíva v tom, že se projde uloženy XML dokument s šablonou, najde se tam voláni funkce „ENTRYPOINT“, to se nahradí voláním funkce „INJECTED“. Za volání funkce „INJECTED“ se vloži seznám volání funkci které odpovídají jednovliým akcim robotu.

5. **Vytvořit realnou reprezentaci plošky.** MAPF Scenario umožňuje zobrazovat plošky na displeji nebo je ukládat do souboru, nebo posílat na tiskárnu. Délka hrany a jeji tlouškta je plně nastavitelná.

6. **Provest testování** Provedéni testování spočíva v rozmistěni robotu na startovní pozice na plošce, jejích spuštěni a zaznemání vysledku.

## 3.2 Testované světy.

Zvolimé si několik ploch s mřižky a rozmistěni robotu pro které budeme provádět test.

1. **Cross.** Na obrázku 3.1 jednoduchy test, kde se protnou cesty dvou robotu.

2. **Plane.** Na obrázku 3.2 Podobný test jako výše, akorat pro tři roboty.

3. **Riddle.** Na obrázku 3.3 je test, který byl inspirován Lloydovou 15. Na malém prostoru jsou zde tři roboti. Zde se předpokláda velké množství rotaci a popojižděni.

4. **Spiral.** Na obrázku 3.4 je zobrazen svět pro testování vlačkového pohyb robotu po spiralovité ploše.

5. **Headless.** Na obrázku 3.5 je zobrazen svět, kde se testuje pohyb dvou robotu po společně uzké cestě.

6. **Basket.** Na obrázku 3.6 je svět, kde se sleduje rozdilna delka akčnich cest dvou robotu. Zde je možné sledovát jaké akce provádi robot, který má kratši akčni cestu, a tim padem si ji musi nějak vyplnit.

7. **Switch.** Na obrázku 3.7 je svět, kde se roboti můsi vyhnout. Testuje se schopnost vyhnout se kolizi.

8. **Unlucky.** Na obrázku 3.8 je svět, kde délka cesy s řešením je výrazně dělši, než součásna pozice robotu od startovní a cílové pozice.

Na obrazcích jsou vyznačené jednotlivé plochy, a v některých vrcholéch jsou umistěny startovní pozice pro robota $i$ s popisem $Si$ a cílova pozice s popiskem $Ci$. Pro plnou definici světa je potřeba navic zvolit také množinu akci, které roboti budou moct provádět. Navic každá akce musí mit přiřazenou nějakou dobu trváni. Tím se budeme zabývat v jednotlivých testech.

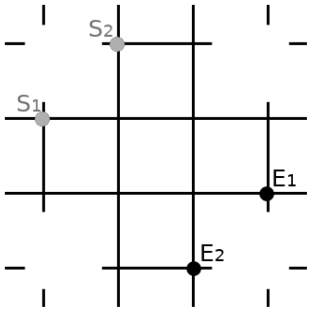
## 3.3 Ozoboti

Pro testování modélu v realnem světe jsme zvolili ozoboty. Ozobot Evo je mály kulaty robot s rozměry přibližně 35 mm v každém směru. Programováni ozobota se provádí ve vyvojovém protředi s názvem OzoBlockly.

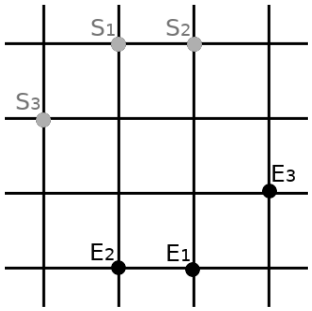
Základní dovédností Eva je jizda a sledování nakreslené čary. Při křížicich se čárach Evo umožňuje si vybrat směr další jizdy, připadně lze zvolit jemnější akce jako rotace. Zde tyto dovednosti se využiji pro pohyb robota po mřižkové ploše. V našich testech se cesty pro agenty převedou na posloupnosti instrukcí pro ozobota Evo. Samotné vykonávaní programu bude spočívat v tom, že ozobot ve vrcholu na plošce přečte instrukci, podle ni zvoli směr další jizdy a sleduje čáru pod ním až do okamžiku kdy dorazi do dalšího vrcholu. Pak čte další instrukce. Podrobnější popis ozobotu se nachází v příloze. [REF Ozka na prilohu 1. ]

## 3.4 Volba prostředi

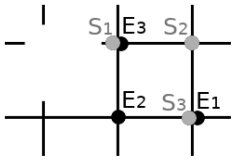
Realný svět se oproti tomu modelovému je o mnoho bohatši. V našem modelovém světě je délka hrany vždy jedna, a přechod z jednoho vrcholu do druhého zabere pravě 1 diskrétni čas. Tato skutečnost ovšem pro testování je nepostačujici. V realném světě se potýkáme s tím že čas je spiše spojity než diskrétni, a změna pozice robota je v čase vždy spojita. Při vyběru prostředi jsme se soustředili na dobu prováděni testu, a to směrem, aby testy zabrali co nejmenší čas.



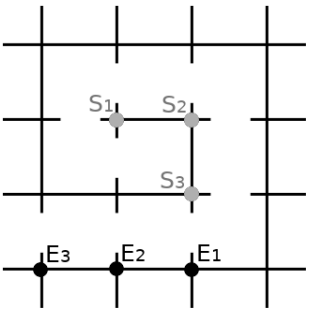

Obrázek 3.1: Svět cross



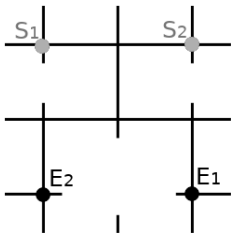

Obrázek 3.2: Svět plane1



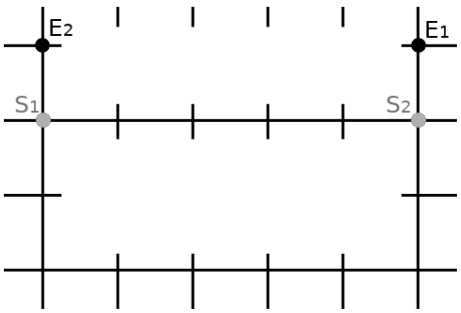

Obrázek 3.3: Svět riddle



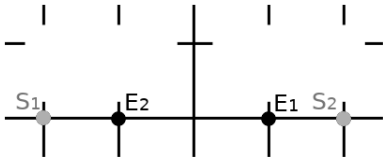

Obrázek 3.4: Svět spiral



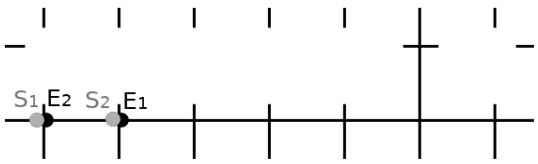

Obrázek 3.5: Svět headless




Obrázek 3.6: Svět basket




Obrázek 3.7: Svět switch




Obrázek 3.8: Svět unlucky

Tedy, délká hrany mezi vrcholy byla zvolena na co nejmenší a současně, aby pro dobrý plán nedocházelo ke kolizim. Délka hrany byla zvolena na 55 mm. Což vychází na 200 pt. Tloušťka hrany byla zvolena na 5 mm, tedy 20 pt, tuto hodnotu doporučuje výrobce ozobotu.

Dále si zavedeme tři skupiny akci pro roboty.

1. **Akce** $A$. Tato skupina akci počita s tím že robot je schopen určit světové směry, a skláda se z akci {jih, server,zapad, vychod,nikam }. Vysledkem těchto akci je posunuti agenta o následujicí vrchol v určeném směru. Jelikož ozoboti nemaji schopnsot určovat světové směry, tak tyto akce nebudeme využivat.

2. **Akce** $B$. Zde množina akci bude vypadat takto { LevoJdi, PravoJdi, VzadJdi, Vpred, Cekej}. Tyto akce, už pro robota nevyžaduji, aby uměl určit světové směry. Akce LevoJdi, PravoJdi vždy natočí robota o 90 stupňu a poté projde po hraně do vrcholu před nim. Akce Vpred přejde do dalšího vrcholu bez otáčeni, a Cekej bude zvolenou dobu čekat v aktualním vrcholu. V programu mapf scenario byla použita anglicka verze: {leftGo, rightGo, waitB, backwardGo, goB}.

3. **Akce** $C$. Tato množina akci vypada takto: { ZatocDoleva, ZatocDoprava,Vpred, Cekej}. Tyto akce jsou rozdělnim akci $B$ na jemnější. Zde se jedna jen o rotaci, pohyb, a čekání. Rotace je opět o 90 stupňů. Tato volba akci se bude hodit pro některou z řešících technik.

Nadále se budeme zabývat akcemi $B$ a $C$. Proto abychom mohli $B$ a $C$ použít, potřebujeme přiřadít každé akci dobu jejího trvání. Zde se opět využijeme požadávek aby akce trváli co nejkratší dobu. Pro každou akci zavedeme jednu z variant. $Real$ a $Uniform$. Akce varianty Real budou takové, které maji nahodně zvolenou dobu běhu. Tyto akce pomohou simulovat dobu běhu v realném prostředi, kdy ve skutku jednotlivé akce zabíraji různou dobu běhu. Varianta $Uniform$ přiřadí každé akci v ramci množiny stejnou dobu běhu. To nam umožni testovát prostředi ve kterém akce robotu maji stejné trváni.

Tabulky 3.1 a 3.2 ukazuji délku trvání akce jak použita v nastaveném prostředi. Poznamenéjme, že skutečné doby trvání akce se řádově o 50 ms různí. Dané rychlosti akci byli zvoleny, jako nejrychlejší proveditelné násobky 500 ms. Vestavěné omezeni ozobota neumožnily akce pro $C$ nastavít na 500ms,a pro $B$ nastavít na 1500 ms. Tedy doba trvání $C$ byla zvolena jako 1000 ms, a doba běhu $B$ jako 2000 ms.

## 3.5 Přehled metod

Zde budeme uvažovat $G_s = (V_s,E_s)$ jako graf mřižkové plochy, $R_s$ roboty na mřižkové ploše. $G_m = (V_m,E_m)$ bude graf a $A_m$ bude seznam agentu, ktery vstupuje do MAPF řešiče. $C$ bude seznam cest, které jsou vystupem MAPF řešíče. $P$ bude seznam planu pro jednotlivé roboty. Budeme předpokládat, že všechny roboti na mřižkové ploše jsou namiřeny na sever. Severem zde budemě rozumět směr, jak je značen na klasickych mapach.

Zavedeme si pomocnou funkci $smer : (u,v) \rightarrow s$, kde $u,v \in V_s, s \in Smery = \{jih, server, zapad, vychod, nikam\}$, ktera říka jakym směrem vede hrana $(u,v)$. Jelikož graf $G_s$ popisuje mřižkovou plochu, tak tato funkce ma smysl.

### 3.5.1 Řešici tehnika Simple

Transformace Svět→MAPF zapišeme jako $SM = (Gf,Vf)$ a bude pro ní plátit, že $Gf$ i $Vf$ bude identita. Tedy graf $G_s = G_m$. Pro každého robota bude plátit, že $Rs_i = s_i$ a $Rc_i = c_i$, tedy startovní i cílova pozice robota je stejna jako startovni a cilova pozice agenta. Nechť *Act* je množína akci, které agent může vykonávat. V stavajici metodě zavedeme *Act* jako množinu { LevoJdi, PravoJdi,VzadJdi, Vpred, Cekej}. Z našeho pohledu to budou dale nedělitelné akce. Roboti si je mohou reprezentovat jako vhodnou sekvenci instrukci.

Jako MAPF řešič zde použijeme DB+SC, tedy ten co optimalizuje dobu běhu a součet cen.

MAPF řešíč vytvoři seznam cest $C$. Cesty převedeme na akce pro agenty

| Real | Trvani akce (ms) | Instrukce |
|---|---|---|
| LevoJdi | 1200 | T68 G68 |
| PravoJdi | 1200 | T68 G68 |
| Vpred | 800 | G68 |
| VzadJdi | 1500 | T71 T71 G73 |
| Cekej | 1000 | - |
| Unif | Trvani akce (ms) | Instrukce |
| LevoJdi | 2000 | T46 G38 |
| PravoJdi | 2000 | T46 G38 |
| Vpred | 2000 | G25 |
| VzadJdi | 2000 | T71 T71 G70 |
| Cekej | 2000 | - |

*Pozn:* TRychlost a GRychlost jsou instrukce pro otoceni a jdi s danou rychlosti.

Tabulka 3.1: Tabulka akci B

| Real | Trvani akce (ms) | Instrukce |
|---|---|---|
| ZatocDoleva | 400 | T68 |
| ZatocDoprava | 400 | T68 |
| Vpred | 800 | G68 |
| Cekej | 600 | - |
| Unif | Trvani akce (ms) | Instrukce |
| ZatocDoleva | 1000 | T21 |
| ZatocDoprava | 1000 | T21 |
| Vpred | 1000 | G54 |
| Cekej | 1000 | - |

*Pozn:* TRychlost a GRychlost jsou instrukce pro otoceni a jdi s danou rychlosti.

Tabulka 3.2: Tabulka akci C

pomoci transformace MAPF→Akce. Nejprve pro agenta $i$ vezmeme cestu $C_i$ = ($v_1$,$v_2$,...,$v_n$). Tuto cestu upravime přidáním předstartovního vrcholu $v_0$ tak, aby $smer((v_0,v_1)) = Sever$. V tomto připadě $G_s = G_m$, a můžeme zde na $G_m$ použit funkci $smer$. Budeme nadále počítat s cestou $Cm_i$ = ($v_0$,$v_1$,...,$v_n$). Poznamenáme, že tento pomocny krok zavadí do cesty $Cm_i$ fakt, že všichni roboti jsou při startu nasměrovani čelem na server. Transformace MA pro agenta $i$ a jeho cestu $C_i$ postupně vezme všechny useky $u_k = usek(C_i,k,3)$ delky 3 a zobrazí je na akce. Samotné zobrazeni na akce by mohlo byt řešeno stavovou tabulkou. Tu si pojmenujeme jako $prechod(u,v)$ a přechody uvedeme v tabulce 3.3. Nechť usek $u_k$ = ($v_1$,$v_2$,$v_3$), pak $směr1 = smer((v_1,v_2))$ a $směr2 = smer((v_2,v_3))$. Pak $prechod(smer1,smer2)$ dá akci pro robota v kroku k. Lze si všimnout, že $smer1$ muze nabyvat hodnoty $Nikam$, což funkce $prechod(u,v)$ neošetřuje. To by se mohlo ošetřít další pomocnou funkci, která by vrácela poslední směr agenta před k-tym směrem $Nikam$. Timto je vytvořena ucelena metoda ktera vytvoři, pro mřižkovou plochu a roboty na ni, seznam akci, které roboty dovedou do cilového stavu.

Tato metoda ovšem je pro mřižkovou plochu a roboty nepoužitelna. Neboť neošetřuje situaci, kde vice agentu při jedné akci zdíleji společnou hranu. Tedy nesplňuje omezni hranovych konfliktu. Stavajici transformace SM a MA tuto vlastnost nepřidávají. Proto tato metoda nebude testovana. Ale bude použita jako základ pro další metody.

### 3.5.2 Řešici tehnika EdgeSplit

Budeme vycházet z předchozí metody, akorat provedeme upravu transformace Svět→MAPF a MAPF→Akce. V předchozi metodě graf $G_s$ určujici mřižkovou plochu byl převeden přesně na graf $G_m$, ktery je vstupem pro MAPF řešíč. V teto metodě provedeme transformaci, ve které každa hrana grafu $G_s$ bude v grafu $G_m$ reprezentovana posloupnosti hrana, vrchol, hrana. Formalně nechť $G_s = (V_s,E_s)$ a $G_m = (V_m,E_m)$. Sice je $G_s$ neorientovany graf, ale budeme ho reprezentovat jako orientovny. Pak pro každou dvojici hran $(u,v) \in G_s$ a $(v,u) \in G_s$ přidame do $G_m$ vrcholy $u,v,x_{\{u,v\}}$. Přidáme také nové hrany $(u,x_{\{u,v\}})$, $(x_{\{u,v\}},v)$, $(v,x_{\{u,v\}})$, $(x_{\{u,v\}},u) \in E_m$. Zavedeme si pro vrcholy grafu $G_m$ nové značeni $x_{\{u,v\}}$ je *od hrany* které značí, že vrchol vzníkl štěpením hrany. A $v$ *od vrcholu* které značí, že k vrchol vznikl k vrcholu původního grafu $G_s$. Abychom zamezili vyměně agentu po společné hraně zavedeme si invariant, že v každem sudem čase jsou všechny agenti ve vrcholu, který je od vrcholu. A v lichych musi byt všichni

| prechod(u,v) | Jih | Sever | Zapad | Vychod | Nikam |
|---|---|---|---|---|---|
| Jih | Vpred | VzadJdi | PravoJdi | LevoJdi | Cekej |
| Server | VzadJdi | Vpred | LevoJdi | PravoJdi | Cekej |
| Zapad | LevoJdi | PravoJdi | Vpred | VzadJdi | Cekej |
| Vychod | PravoJdi | LevoJdi | VzadJdi | Vpred | Cekej |

*Pozn:* první sloupec udáva predchozi stav, první řádek udáva následujici stav.

Tabulka 3.3: Tabulka určuje funkci, ktera z aktualního kroku agenta počita jeho akci

agenti ve vrcholéch které jsou od hrany. Nastáva zde problém, s reflexivními hranámi. Neboď v čase $t$ sudé je agent a v vrcholu $v$ a v čase $t+1$ liche agent může zůstat v stejném vrcholu. Tim je invariant porušen. Proto graf $G_m$ nebude obsahovat reflexivni hrány. Tedy $(x,x) \notin E_m \forall v \in V_m$ . Jelikož jsme pro každou hranu $(u,v) \in E_s$ vytvořili hrany $(v, x\{u,v\}) \in E_m$ a $(x\{u,v\},v \in E_m)$, tim jsem si zajistili reflexivni hranu pro vrchol $v$ která nam invariant neporuši. Situaci kdy vrchol nemá žadného souseda můžeme ošetřít ták, že jeho původní reflexifni hránu rozdělime podobně jako hrany mezi dvěma vrcholy.

Transofmaci MAPF→Akce upravíme oprotí předchozí verze tak, že nejdřivé odfiltrujeme všechny vrcholy od hrany. Tedy pro agenta $i$ a jeho cestu $C_i = (v_0, x_{\{0,2\}}, v_2, x_{\{2,3\}}, ..., v_n)$ vznikne nová cesta $Cm_i = (v_0, v_2, ..., v_n)$. Novou cestu lze získat buď tím, že smažeme všechny liché vrcholy na cestě, a nebo smažeme všechny vrcholy od hrany. Upravenou cestu necháme zpracovat transformaci MAPF→Akce půdovní Jednoduché metody.

### 3.5.3 Řešici tehnika EdgeSplitDisj

Zde opět vyjdeme z řešíci techniky Simple, a použijeme i jeji SM a MA transformace, akorat použijeme MAPF řešič bez hranových konfliktu. Vysledna metoda by měla byt totožna s metodou s děleným hran. Rozdil by se mohl projevit v časech potřebnych pro tvorbu planu.

### 3.5.4 Řešici tehnika VertexSplit

Doposud jsme brali jako základní množinu akci robota množinu { LevoJdi, PravoJdi,VzadJdi, Vpred, Cekej}. Zde si uvedeme jinou množinu základních akci robota a to { ZatocDoleva, ZatocDoprava,Vpred, Cekej}. Je vidět, že tato množína akci je trošku menší než předchzi. Jeji akce jsou jemnější, a lze jimi popsat předchozí množinu akci. Jemnější akce nam umožní tvořit vice detailni plány pro agenty. A budou také vyžadovat složitější transformace Svět→MAPF a MAPF→Akce.

Transoformaci Svět→MAPF si zavedeme nasledujicím způsbobem. Akce ZatocDoleva, a ZatocDoprava neměni vrchol $v \in V_s$ jen měni natočeni směru, ve kterém robot zrovna je. Budeme ve vrcholu vest informace o čtyřech základní natočeni $Natoceni = \{sever, vychod, jih, zapad\}$. Abychom tuto skutečnost zachytili v grafu $G_m$ provedeme štěpeni $v \in V_s$ na čtyří vrcholy $v_{vychod}, v_{zapad}, v_{sever}, v_{jih} \in V_m$. Do $E_m$ přidáme hrany $(v_{vychod}, v_{sever})$, $(v_{sever}, v_{zapad})$, $(v_{zapad}, v_{jih})$, $(v_{jih}, v_{vychod})$ a podobně hrany v opačném směru. Přidané hrany budou reprezentovat jednotlivé rotace agenta ve vrcholu. Zavedeme si též hrany pro přechod mezi původnimi vrcholy. Pro hranu $(u,v) \in E_s$ podívame se na $s = směr(u,v)$ viz 3.5 přídáme do $E_m$ hranu: $(v_s, u_s)$. Neboli, v jakém směru dana hrana bude, ty směry taky hrana propoji. Abychom potom byli schopni z grafu $G_m$ určit, jake vrcholy reprezentuji rotace a jaké reprezentuji přesun po hraně v původním grafu, zavedeme si pro kažzdy vrchol $v \in V_m$ zobrazeni $vzd : V_m \to V_s$. Toto zobrazeni nám bude vracet vrchol, který byl původce vstupního vrcholu při štěpeni. Tedy $vzd(v_x) = v$, kde $x \in Natoceni$. Pak pro dva vrcholy $u,v \in V_m$ pokud $vzd(u) \neq vzd(v)$ tak víme že dané vrcholy reprezentuji v původním grafu různé vrcholy. Zavedeme si taky zobrazeni $smerJ : V_m \to Natoceni$ timto způsobem $smerJ(v_x) = x$. Toto zobrazeni

nam pak pomůže určit o jakou rotaci se jedna.

Jelikož v grafu $G_m$ hrany už nereprezentuji jen přechod mezi vrcholy v $G_s$ ale i rotace, tak je potřeba upravit MAPF řešič, aby vytvářel jiná omezni, vhodná pro graf $G_m$. Chceme aby ve vrcholu v nehledě na rotací byl maximalně 1 agent. Nechť $B[T,A,V]$ bude pole používané v mapf řešiči. Všechny vrcholy v $G_m$ si rozložim na třidy ekvivalence $TEQ$ podle $vzd$.Pak pro každého agenta A, pro každy čas T a pro každou množinu vrcholu $V_e \in TEQ$ plati, že $\sum_{V \in V_e} B[T,A,V] \leq 1$. Podminku omezeni hranových konfliktu neni potřeba provádět pro všechny hrany grafu $G_m$, ale jen pro konkretní množinu dvojic hran. A to pro hrany $(a,b)$ a $(c,d), a,b,c,d \in G_m$ pro které plati, že $vzd(a),vzd(b) = vzd(c), vzd(d)$, , tedy že dvojice hrán $(a,b)$ a $(c,d)$ v původním grafu $G_s$ představuje přechod po jedné hraně v opačnych směrech. Kod ktery v pikatu zamezi hranove konflikty by mohl vypadat nasledovně:

```
//$ pro kazdou dvojici hran v DisjointE,
// zakazat pouziti hran naraz
foreach(T in 1..M, VSet in DisjointE, A in 1..K)
        VSet = $ep($e(Vs,Ve),$e(Us,Ue)),
        foreach(C in 1..K)
                if A != C then
                        (B[T,A,Vs] #/\ B[T+1,A,Ve])  #=>
                                B[T,C,Us] + B[T+1,C,Ue] #=< 1
                end
        end
end,
```

$DisjointE$ představuje množinu všech dvojic hran, u kterých chci zamezi použtí obou hrán mezi časem $T$ a $T+1$ pro každou dvojici agentu. Samotné omezeni pák říka: Pro každou dvojici hrán $(Vs,Ve)$, $(Us,Ue)$ Pokud agent A je v čase T na pozici Vs a v čase $T+1$ na pozici $Ve$, tak žadny jiny agent nesmi v čase $T$ byt v vrcholu $Us$, a v čase $T+1$ v čase $Ue$.

Transformaci MAPF→Akce provedeme nasledujicím způsobem. Mějme cestu $C_i$, ktera bude vysledkem MAPF řešiče. Pro každ $u_k = usek(C_i,k,2)$ otestujeme zda je dana akce rotace, přechod po hraně a nebo čekáni. Nechť $u_k = (v_1,v_2)$. Pokud $vzd(v_1) = vzd(v_2)$ tak se jedná o rotaci. A akce je pak určena funkci, ktera je popsana v tabulce 3.4. Pokud se jedna o přechod mezi vrcholy tak vysledkem je akce Vpred.

| rot(u,v) | Jih | Sever | Zapad | Vychod |
|---|---|---|---|---|
| Jih | Cekej | – | ZatocDoprava | ZatocDoleva |
| Server | – | Cekej | ZatocDoleva | ZatocDoprava |
| Zapad | ZatocDoleva | ZatocDoprava | Cekej | – |
| Vychod | ZatocDoprava | ZatocDoleva | – | Cekej |

*Pozn:* první sloupec udáva predchozi stav, první řádek udáva následujici stav.

Tabulka 3.4: Tabulka určuje funkci, ktera prechod hrany (u,v) prevadi na rotaci

### 3.5.5 Řešici tehnika EdgeSplitRob

Tato tecnika je rozšířením půdnodni EdgeSplit o podmínku 1-robustnosti. Vnitřně je přidana podmínka pro 2-robustnost. 2-robustnost, proto že EdgeSplit zjemňuje graf, a vrcholy původního grafu maji v grafu zjemněném mezi sebou dva krát větši delky cest.

### 3.5.6 Řešici tehnika VertexSplitRob

Tato tecnika je rozšířením půdnodni VertexSplit o podmínku 1-robustnosti. Zde rozumíme 1-robůtnost vůči původnímu grafu. Jelikož vstupní graf $G_s$ po provedeni transformace Svět→Map byl hodně změněn nelze použit podmínku pro k-robustnost v základní variantě. Zvolíme variantu 1-robustnosti. která říka, že pokud agent $A$ byl ve vrcholu $V$ v čase $T$, tak v čase $T-1$ nesměl byt žadny jiny agent ve vrcholech reprezentujicich rotaci daného vrcholu. Vezměme $TEQ$ jako rozklad vrcholu VertexSplit na třidy ekvivalence podle $vzd$. Nasledujici kod zachycuje tuto podminku.

```
%pokud je agent A v case T na pozici V, tak
%  tak zadny jiny agent nebyl v case T-L az T na pozici V
foreach(T in 1..M,A in 1..K, V in 1..N)
        getCompanion(V,DisjointV,VertNeibs),
        B[T,A,V] #=> sum([B[Prev,A2,VN] : A2 in 1..K, A2!=A,
                        Prev in max(1,T-1)..T,VN in VertNeibs]) #= 0
end,
```

# 4. Experimenty

Cílem experimentu je porovnát rozdíl mezi simulovanou dobou běhu a dobou běhu skutečnou. A tím určit kvalitu simulatoru. Programem jsme si vygenerováli základni sadu map, a pozice pro roboty. Předmětem testovani jsou čtyři řešici techniky: EdgeSplit, VertexSplit, EdgeSplitRob VertexSplitRob. a jejích varianty Real a Unif. Pro vzniklých 8 kombinaci provedeme testování pro každou mapu 5 krát. A zaznamenáme dobu běhu. Vysledky experimentu byli zaznamenáne v tabulce 4.1. Podrobnější vysledky jsou v příloze. Z tabulky je vidět, že varianta Real EdgeSplit ukazuje nejmenši počet doběhnuti do konce. Naopak Robustni varianty doběhly prakticky bez chyby. Z tabulky z přiloze lze vysledovát průměrna odchylka uniform simulace od jeho testu byla jen 0,3 sekundy. Kdežto odchýlka ve variatně real čini 1,5 sekund. Obě hodnoty považujeme za dobré vysledky.

| Var | Technika | DiffAvg | RunAvg | ColAvg | WarnAvg |
|---|---|---|---|---|---|
| REAL | ES | 2,69 | 2,63 | 1,63 | 0,00 |
| REAL | VS | 1,22 | 4,50 | 3,88 | 1,13 |
| REAL | ESR | 1,21 | 5,00 | 0,00 | 0,00 |
| REAL | VSR | 0,96 | 4,88 | 0,00 | 0,00 |
| UNIF | ES | 0,69 | 4,50 | 4,25 | 1,25 |
| UNIF | VS | 0,15 | 4,88 | 1,00 | 1,88 |
| UNIF | ESR | 0,27 | 5,00 | 0,00 | 0,00 |
| UNIF | VSR | 0,12 | 5,00 | 0,00 | 0,00 |

*Pozn:* ES - EdgeSplit, VS - VertexSplit, xR - Robust, (vše v sekundach)
DiffAvg - průměrné rozdily simulováného času od realného,
RunAvg - průměrný počet úspěchu,
RunAvg - průměrný počet kolizi,
RunAvg - průměrný počet varováni ( skoro ke kolizi došlo )

Tabulka 4.1: Výsledky testu

# Závěr

Program fungoval jak měl, až na drobné záhady všechny testy byli provedené úspěšně. A experimenty ukázali odchylka doby běhu simulace od reálné hodnoty je řádově do několika málo sekund. Také se vysledovalo že varianta EdgeSplit Real měla přibližně něco přes polovinu uspěšných běhu. Zvolením varianty na Uniform se dosáhlo většího počtu úspěchu, nicméně průměrný počet kolizi je stále veliký. Varianty algoritmu s Robustnosti naopak vykazovali prakticky nulovou chybovost. Hodně záleží na skutečné rychlosti robota. Když se zjistí přesná rychlost robota, umožňuje to lepší plánováni. Ozoboti se zdají byt poměrně málo náchylné na rozdíly v rychlosti jízdy a to nehledě na stav baterie. A celkem se ukazuji jako dobra varianta na testováni.

# Seznam použité literatury


Barták, R., Zhou, N.-F., Stern, R., Boyarski, E. a Surynek, P. (2017). Modeling and solving the multi-agent pathfinding problem in picat. In *Tools with Artificial Intelligence (ICTAI), 2017 IEEE 29th International Conference on*, pages 959–966. IEEE.

Surynek, P. (2012). On propositional encodings of cooperative path-finding. In *Tools with Artificial Intelligence (ICTAI), 2012 IEEE 24th International Conference on*, volume 1, pages 524–531. IEEE.

Surynek, P., Švancara, J., Felner, A. a Boyarski, E. (2017). Variants of independence detection in sat-based optimal multi-agent path finding. In *International Conference on Agents and Artificial Intelligence*, pages 116–136. Springer.

Yu, J. a LaValle, S. M. (2013). Planning optimal paths for multiple robots on graphs. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3612–3617. IEEE.

Zhou, N.-F., Kjellerstrand, H. a Fruhman, J. (2015). *Constraint solving and planning with picat*. Springer.

# A. Přílohy

## A.1 Ozoboti a Ozoblockly.

Ozobot Evo je malý kulatý robot s rozměry přibližně 35 mm v každém směru. Programováni ozobota se provádí ve vývojovém prostředí s názvem OzoBlockly.

### A.1.1 Ozobot Evo.

Schopnosti ozobota by se dali rozdělit na několik kategorii:

1. **Senzory.** Evo má vespod 7 senzoru na detekci intenzity světla, které používá pro detekcí čáry pod sebou. Dále má vespod uprostřed jeden senzor pro detekci barvy. Jsme schopni programově rozlišit základních 8 barev a to: červenou, žlutou, zelenou, růžovou, modrou, světlemodrou(azurovou), bílou a černou. Neboli všechny kombinace základních třech barev.

   Dále ozobot Evo má čtyři senzory na detekci překážek. Dva jsou ve předu blíže k bokům, a dva jsou vzadu také blíže k bokům. maximální vzdálenost detekce překážky je přibližně 5cm.

2. **Signalizace.** Evo má k dispozici celkem 6 barevných diod, které podporuji zobrazení barev v rozsahu 0-127 pro složku. Jedná dioda je ve středu při pohledu ze shora. Zbývajících pět jsou umístěny v řadě za sebou v přední části ozobota. Evo má také vestavěny reproduktor. Vývojové prostředí umožňuje přehrát tón jistě výšky, říci některou barvu, číslo, a nebo některé z dalších vestavěných slov. Třeba „ha ha ha“.

3. **Pohyb.** Základní a nejdůležitější schopnosti Eva zůstává pohyb. Evo je vybaveno dvěma motory a dvěma kolečky, které umožňuji pohyb dopředu dozadu a rotaci. Každé kolečko je možné ovládat zvlášť. Je možné také nastavovat rychlost otáčeni koleček. V praxi se ukázalo, že robot je odladěn pro pohyb rychlosti 30mm/s. Při rychlostí, která je výrazně pomalejší robot projevoval defekty v plynulosti pohybu a zatčeni.

   Vývojové prostředí pro ozobota neobsahuje instrukce pro čteni jednotlivých senzoru pro detekci čáry. Pro detekcí a sledování čáry nabízí jen základní rozhraní těmito příkazy:

   (a) Jeď po čáře do dalšího průsečíku a nebo do konce čáry.

   (b) Výběr směr další jízdy.

   (c) Nastav rychlost jízdy při sledováni čáry.

   (d) Zjistí barvu konce čáry nebo průsečíku, ve kterém jsi.

   Doporučena šířka čáry, kterou ozobot dokáže sledovat je 5mm. Při užších či širších čárách ozobot má problém čáru rozpoznat. Je dobré poznamenat, že čáru která zatáčí do pravého úhlu chápe ozobot jako čáru bez průsečíku a tedy v rohu příkaz pro sledování čáry neskonči. Z toho důvodu všechny navržené plochy mají po stranách doplňující čáry. Nejlepších výsledku při

sledování čáry se dosáhlo s potištěným papírem na laserové tiskárně, kde barva čáry byla černa. Použitelná byla i lihový fix na papír. Naopak nerovné povrchy působí pro rozpoznávaní čar potíže. Rozpoznávaní čar funguje taky na led display, bylo testováno na displeji Asus VN274.

4. **Komunikace a programování.** Evo disponuje bluetooth modulem, a má k dispozici mikro usb vstup. Příjmy přistup pro Ozobota není přes usb ani bluetooth zpřístupněn. Programovat se dá pouze s pomoci OzoBlockly.

Příkaz „Jeď po čáře do dalšího průsečíku a nebo do konce čáry“ ma blokující charakter. Jakmile je příkaz spuštěn, tak blokuje následující příkazy, a to až do okamžiku, než příkaz skončí. Naopak příkaz „nastav rychlost koleček“ je neblokující. A umožňuje vykonávat jiné příkazy během pohybu robota. Tedy po dobu sledování čáry není robot schopen provádět jiné instrukce, a to až do okamžiku kdy dojede do konce čáry nebo do průsečíku s jinou čarou. Tato vlastnost znemožňuje během pohybu robota po čáře, detekovat kolize s jinými roboty. Pokud bychom chtěli detekovat srážky robotu, bylo by potřeba ručně naprogramovat sledování čáry a vkládat mezi něj úseky pro čteni senzoru pro detekci vzdálenosti. Zde ovšem narážíme na to že, že těch 7 senzoru určených pro detekci a následování hrany nejsou v stavající verzi OzoBlockly dostupné. Způsob jak to lze částečně obejit je ručně vykonávat rotaci a slepě se pohybovat dopředu ignorujíc čáru pod robotem, a v po jisté době, třeba v polovině hrany začít opět sledovat čáru. Jinak současná verze OzoBlockly nenabízí příkazy pro jednoduché sledovaní čáry a současné vykonávaní jiných příkazu. [ Dokumentace Ozoblockly Linefollowing ].

### A.1.2 Ozoblockly.

Evo je snadno programovatelné. Ozobot Evo je od základu postaven jako hráčka a výuková pomůcka, pro učeni děti logickému myšleni a programováni. Tomu odpovídá i vývojové prostředí OzoBlockly. OzoBlockly vychází z open source knihovny s názvem Blockly. Blockly je JavaScriptová knihovna, která běží na straně klienta.

Programování v OzoBlockly se převážně skládá z vytvářeni a skládaní jednotlivých bloku do celku. Jeden blok může byt jedna nebo více instrukci, to záleží na konkretní abstrakci. V současností OzoBlockly slouží k programování pro dva typy robotu a to Bit a Evo. Bit je o poznání jednodušší. Vývojové studio nabízí celkem 5 úrovni zapouzdřeni instrukci pro ozobota. A to od nejjednodušší, až po nejsložitější. V nejsložitější úrovní nabízí práci s proměnnými, umožňuje vytvářet funkce, obsahuje instrukce pro aritmetiku a logické operátory. Taktéž dovoluje větveni, cykly a prací s polem. Samozřejmosti jsou instrukce pro ovládaní kol, diod, vydávaní zvuku, vypnutí ozobota, sledování čár a čteni hodnot senzoru pro detekci překážek. Používá zásobníkové volání funkci, a umožňuje psát rekurzivní programy. Celka velikost programu je omezena na 2048Bytu. Dá se říci že ozoblockly je plnohodnotný programovací jazyk.

### A.1.3 Nahrávaní kódu.

Ozoblocklý je vývojové prostředí, založené na JavaScriptu, a běží v prohlížeči na straně klienta. Je dostupné z internetové stránky ozoblockly.com.

Nahrávaní kódu do ozobotu Evo lze provést dvěma způsoby:

1. **Pomoci světelných signálu na obrazovce.** U Ozobota Evo lze při zapínaní podržet zapínací tlačítko a tím ho přepnout do programovacího režimu. Následně ho přiložit na vyznačené místo na monitoru v ozoBlockly. A stisknutím tlačítka „Load“ započne nahrávaní kódu pro ozobota. Program pak lze spustit dvojklikem na spouštěcí tlačítko. Nevýhoda této metody je, že pro větší programy, obzvlášť ty s zvukovými příkazy, je nahrávaní kódu pomalé. Třeba program, který má zvukové instrukce zabírá kolem 1317Bytu, trvá nahrát přibližně 3 minuty a 21 sekundy. Pro reálné testování se táto varianta stává nepraktickou.

2. **Pomoci aplikace ozobot Evo v androidu nebo IOSu.** Tato varianta zahrnuje mezičlánek mezi ozobotem a ozoblockly. Takový mezičlánek může byt tablet nebo telefon s přístupem k internetu a funkčním bluetooth. Nejdříve do mezičlánku stáhneme aplikaci Ozobot Evo, a v něm si založíme účet. K učtu je vázáno 12 pozic pro úkladaní programu v cloudu. Pak se přihlásíme do ozoblockly pod daným účtem. Následně si vytvořme program a uložíme ho do jedné z pozic. Aplikace má přistup k uloženým pozicím s programy. Pak spárujeme robota s mezičlánkem, a následně spustíme program ze zvolené pozice.

   Pokud už je mezičlánek aktivní a funkční, nahrávaní pak probíhá přes uloženi do pozice tlačítkem „Save“ v ozoblockly. A následně kliknutím v aplikaci na pozici s uloženým programem, proběhne nahraní programu do ozobota. Tato varianta je složitější na zprovozněni a pochopeni. Vyžaduje mezičlánek, založený účet, a funkčné internet. Nabízí ale rychlejší nahrávaní. A nezanedbatelnou výhodou je možnost spouštět jeden program ve vice robotech v stejný okamžik.

Ozoblockly umožňuje vytvořené programy také ukládat do souboru, a ze souboru je načítat. Soubor je ve formatu xml.

## A.2 Vysledky experimentu

| Varianta | Řešíč | No. | mkt | mkr | diff | suc | col | wrn |
|---|---|---|---|---|---|---|---|---|
| 1.cross | | | | | | | | |
| REAL | ES | 5 | 6,7 | 8,8 | 2,1 | 3 | 3 | 0 |
| REAL | VS | 6 | 5,8 | 6,8 | 1,0 | 3 | 3 | 0 |
| REAL | ESR | 7 | 9,6 | 9,3 | 0,3 | 5 | 0 | 0 |
| REAL | VSR | 8 | 7,7 | 7,3 | 0,4 | 5 | 0 | 0 |
| UNIF | ES | 9 | 11 | 11,3 | 0,3 | 5 | 0 | 0 |
| UNIF | VS | 10 | 9 | 9,0 | 0,0 | 5 | 0 | 0 |
| UNIF | ESR | 11 | 13 | 13,7 | 0,7 | 5 | 0 | 0 |
| UNIF | VSR | 12 | 10 | 10,1 | 0,1 | 5 | 0 | 0 |
| 2.plane | | | | | | | | |
| REAL | ES | 13 | 6,7 | 9,0 | 2,3 | 1 | 4 | 0 |
| REAL | VS | 14 | 6,4 | 8,0 | 1,6 | 5 | 13 | 2 |
| REAL | ESR | 15 | 7,4 | 9,0 | 1,6 | 5 | 0 | 0 |
| REAL | VSR | 16 | 7,6 | 8,8 | 1,2 | 5 | 0 | 0 |
| UNIF | ES | 17 | 11 | 11,8 | 0,8 | 4 | 2 | 0 |
| UNIF | VS | 18 | 10 | 10,1 | 0,1 | 5 | 2 | 2 |
| UNIF | ESR | 19 | 13 | 13,2 | 0,2 | 5 | 0 | 0 |
| UNIF | VSR | 20 | 12 | 11,8 | 0,2 | 5 | 0 | 0 |
| 3.riddle | | | | | | | | |
| REAL | ES | 21 | 3,7 | 0,0 | 3,7 | 0 | 0 | 0 |
| REAL | VS | 22 | 5,2 | 6,0 | 0,8 | 5 | 0 | 2 |
| REAL | ESR | 23 | 7,7 | 8,6 | 0,9 | 5 | 0 | 0 |
| REAL | VSR | 24 | 6,4 | 6,9 | 0,5 | 5 | 0 | 0 |
| UNIF | ES | 25 | 5 | 7,5 | 2,5 | 5 | 17 | 0 |
| UNIF | VS | 26 | 8 | 8,0 | 0,0 | 5 | 0 | 0 |
| UNIF | ESR | 27 | 13 | 13,4 | 0,4 | 5 | 0 | 0 |
| UNIF | VSR | 28 | 10 | 10,2 | 0,2 | 5 | 0 | 0 |
| 4.spiral | | | | | | | | |
| REAL | ES | 29 | 14,2 | 16,6 | 2,4 | 2 | 2 | 0 |
| REAL | VS | 30 | 15,6 | 17,2 | 1,6 | 5 | 4 | 5 |
| REAL | ESR | 31 | 16,2 | 18,1 | 1,9 | 5 | 0 | 0 |
| REAL | VSR | 32 | 16,2 | 17,8 | 1,6 | 4 | 0 | 0 |
| UNIF | ES | 33 | 27 | 27,6 | 0,6 | 2 | 10 | 5 |
| UNIF | VS | 34 | 23 | 23,1 | 0,1 | 5 | 0 | 9 |
| UNIF | ESR | 35 | 31 | 31,0 | 0,0 | 5 | 0 | 0 |
| UNIF | VSR | 36 | 24 | 23,9 | 0,1 | 5 | 0 | 0 |

*Pozn:* ES - EdgeSplit, VS - VertexSplit, xR - Robust, mkt - Makespan simulovany, mkr - Makespan reálný, suc - početu spěšných běhu,
col - počet kolizi, wrn - počet varování

| Varianta | Řešíč | No. | mkt | mkr | diff | suc | col | wrn |
|---|---|---|---|---|---|---|---|---|
| 5.headless | | | | | | | | |
| REAL | ES | 37 | 6,8 | 0,0 | 6,8 | 0 | 0 | 0 |
| REAL | VS | 38 | 7,8 | 9,2 | 1,4 | 3 | 6 | 0 |
| REAL | ESR | 39 | 7,8 | 7,3 | 0,5 | 5 | 0 | 0 |
| REAL | VSR | 40 | 8,4 | 9,4 | 1,0 | 5 | 0 | 0 |
| UNIF | ES | 41 | 11 | 11,4 | 0,4 | 5 | 2 | 3 |
| UNIF | VS | 42 | 13 | 13,2 | 0,2 | 5 | 4 | 2 |
| UNIF | ESR | 43 | 13 | 13,5 | 0,5 | 5 | 0 | 0 |
| UNIF | VSR | 44 | 14 | 13,8 | 0,2 | 5 | 0 | 0 |
| 6.basket | | | | | | | | |
| REAL | ES | 45 | 10,6 | 11,9 | 1,3 | 5 | 0 | 0 |
| REAL | VS | 46 | 10,6 | 12,1 | 1,5 | 5 | 0 | 0 |
| REAL | ESR | 47 | 10,6 | 12,2 | 1,6 | 5 | 0 | 0 |
| REAL | VSR | 48 | 10,6 | 11,7 | 1,1 | 5 | 0 | 0 |
| UNIF | ES | 49 | 21 | 21,3 | 0,3 | 5 | 0 | 0 |
| UNIF | VS | 50 | 15 | 15,3 | 0,3 | 5 | 0 | 0 |
| UNIF | ESR | 51 | 21 | 21,1 | 0,1 | 5 | 0 | 0 |
| UNIF | VSR | 52 | 15 | 15,2 | 0,2 | 5 | 0 | 0 |
| 7.switch | | | | | | | | |
| REAL | ES | 53 | 7,7 | 9,2 | 1,5 | 5 | 4 | 0 |
| REAL | VS | 54 | 8,6 | 9,4 | 0,8 | 5 | 5 | 0 |
| REAL | ESR | 55 | 9,7 | 11,1 | 1,4 | 5 | 0 | 0 |
| REAL | VSR | 56 | 8,2 | 9,2 | 1,0 | 5 | 0 | 0 |
| UNIF | ES | 57 | 13 | 13,6 | 0,6 | 5 | 3 | 2 |
| UNIF | VS | 58 | 13 | 13,3 | 0,3 | 5 | 2 | 2 |
| UNIF | ESR | 59 | 17 | 17,3 | 0,3 | 5 | 0 | 0 |
| UNIF | VSR | 60 | 13 | 13,0 | 0,0 | 5 | 0 | 0 |
| 8.unlucky | | | | | | | | |
| REAL | ES | 61 | 11,7 | 13,1 | 1,4 | 5 | 0 | 0 |
| REAL | VS | 62 | 12,6 | 13,6 | 1,0 | 5 | 0 | 0 |
| REAL | ESR | 63 | 14,7 | 16,1 | 1,4 | 5 | 0 | 0 |
| REAL | VSR | 64 | 13,2 | 14,1 | 0,9 | 5 | 0 | 0 |
| UNIF | ES | 65 | 23 | 23,0 | 0,0 | 5 | 0 | 0 |
| UNIF | VS | 66 | 18 | 17,8 | 0,2 | 4 | 0 | 0 |
| UNIF | ESR | 67 | 29 | 29,0 | 0,0 | 5 | 0 | 0 |
| UNIF | VSR | 68 | 19 | 19,0 | 0,0 | 5 | 0 | 0 |

*Pozn:* ES - EdgeSplit, VS - VertexSplit, xR - Robust, mkt - Makespan simulovany, mkr - Makespan reálný, suc - početu spěšných běhu,
col - počet kolizi, wrn - počet varování