

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

PLÁNOVÁNÍ CESTY ROBOTA POMOCÍ DYNAMICKÉHO PROGRAMOVÁNÍ

ROBOT PATH PLANNING BY MEANS OF DYNAMIC PROGRAMMING

DIPLOMOVÁ PRÁCE
DIPLOMA THESIS

AUTOR PRÁCE **Bc. IVO STÁREK**
AUTHOR

VEDOUCÍ PRÁCE **RNDr. JIŘÍ DVOŘÁK, CSc.**
SUPERVISOR

BRNO 2009

ZADÁNÍ ZÁVĚREČNÉ PRÁCE

(na místo tohoto listu vložte originál a nebo kopii zadání Vaš práce)

ABSTRAKT

Práce je věnována plánování cesty robota pomocí dynamického programování v diskrétním prostředí. Teoretická část se věnuje současné situaci v tomto oboru a principu aplikace Markovova rozhodovacího procesu na oblast plánování cesty. Praktická část se věnuje vlastní implementaci dvou algoritmů využívajcích principů MRP.

ABSTRACT

This work is dedicated to robot path planning with using principles of dynamic programing in discrete state space. Theoretical part is dedicated to actual situation in this field and to principle of applying Markov decission process to path planning. Practical part is dedicated to implementation of two algorithms based on MDP principles.

KLÍČOVÁ SLOVA

Plánování cesty robota, dynamické programování, Markovův rozhodovací proces.

KEYWORDS

Robot path planning, dynamic programing, Markov decision process.

PODĚKOVÁNÍ

Rád bych poděkoval RNDr. Jiřímu Dvořákovi za jeho podněty a poznatky, které mi pomohly k vyhotovení této práce.

Obsah:

Zadání závěrečné práce....3
Abstrakt....5
Poděkování....7
1 Úvod....11
2 Metody použité v plánování cesty robota....13
2.1 Úvod....13
2.1.1 Základní složky plánování....13
2.1.2 Způsoby využití plánů....14
2.2 Metody založené na dekompozici buněk....15
2.2.1 Přesná dekompozice buněk....15
2.2.2 Metody přibližné dekompozice buňek....16
2.3 Heuristické metody plánování....17
2.3.1 Základní algoritmy....17
2.3.2 Přírůstkové replánující algoritmy....18
2.3.3 Anytime algoritmy....19
2.3.4 Anytime replánující algoritmy....21
2.4 Hiearchické plánování cesty....22
2.5 Umělá potenciálová pole....23
2.6 Plánování za užití genetických algoritmů....25
2.7 Metody založené na cestovní mapě....26
2.7.1 Voroného diagramy....26
2.7.2 Graf viditelnosti....27
2.8 Plánování cesty pomocí neuronových sítí....27
3 Dynamické programování v plánování cesty robota a jeho aplikace....31
3.1 Markovské rozhodovací procesy pro plánování cesty....31
3.1.1 Výpočet optimální strategie a hodnoty v MRP....32
3.1.2 Metody pro efektivnější výpočty MRP....34
3.2 Algoritmy SDP a LAO* a jejich aplikace....35
3.2.1 Směrované dynamické programování....35
3.2.2 Algoritmus LAO* a jeho modifikace....36
3.2.3 Hlavní části algoritmu SDP....38
3.2.4 Hlavní části modifikovaného algoritmu LAO*....39
4 Implementace navrženého řešení....41
4.1 Zpracovaný program....41
4.2 Užité algoritmy....41
4.3 Program a jeho ovládání....41
4.3.1 Hlavní části aplikace....42
5 Testy a srovnání algoritmů....49
6 Závěr....57
7 Seznam použité literatury....59

1 ÚVOD

Ve své práci se zabývám plánováním cesty robota. Oblast robotiky a aplikace robotů přináší mnoho výhod. Především to je schopnost robotu jednat samostatně v různých prostředích. Známou aplikací tohoto typu je japříklad robot Mars Rover jehož vlastnosti se osvědčily v takových podmínkách jako je průzkum Marsu. Jeho výdrž a schopnosti adaptovat se na místní prostředí několikanásobně vyvážily náklady, které by byly vynaloženy na řešení průzkumu cizího vesmírného tělesa. Úspěch robotického vozítka umožňil vývoj mnohem dokonalejších a sofistikovanějších robotů, které budou užity k dalšímu průzkumu sluneční soustavy. Vlastnost robotů pracovat v cizím nebo agresivním prostředí otevírá široké možnosti aplikace a náhrady robotů za lidskou práci.

V umělé inteligenci se pod pojmem plánování rozumí vyhledání sekvence logických operací nebo akcí, která transformuje počáteční pozici stavu v prostředí do požadovaného cílového stavu. Základní potřebou v robotice je dostatek algoritmů převádějících globální úlohy zadané člověkem (například vyjet s autem z garáže) do lokálních, nízkoúrovňových sekvencí popisů nebo instrukcí rozdělujících složitou úlohu do sekvencí jednoduchých akcí. Plánování cesty robota se zabývá generováním vhodné cesty, aniž by se robot setkal s překážkami. Aplikace plánování nalézají místo v oblastech jako průmyslová robotika, počítačová grafika, návrh struktury léků nebo automatický průzkum prostoru. Algoritmů pro plánování cesty existuje celá řada. Ve své práci jsem se pokusil zpracovat dvě metody plánování. Obě mají společné vlastnosti: jedná se o metody heuristické a jsou založeny na optimalizaci prostupu prostorem pomocí dynamického programování. Užití algoritmů dynamického programování zefektivňuje výpočet a k potřebnému řešení se dochází rychleji. Součástí práce je také základní souhrn a analýza nyní používaných metod pro plánovaní cesty.

Cílem mé diplomové práce bylo analyzovat dosavadní přístupy v plánování cesty robota, navrhnout a implementovat systém pro plánování cesty robota využívající dynamické programování a provést srovnávací a ověřovací experimenty.

2 METODY POUŽITÉ V PLÁNOVÁNÍ CESTY ROBOTA

Plánování cesty začítná tím, že robot dostane mapu prostoru, polohu startovního stavu a polohu cílového stavu. Schopnost plánování závisí na výpočtu nebo na naučení se optimální cesty k cíli.

2.1 Úvod

Plánovacích algoritmů existuje celá řada, stejně tak existuje množství jejich skupin a možností jejich rozdělení. K nejzákladnějším rozdělením patří rozdělení podle charakteru prostředí obklopujícího robota a podle modelu obcházení překážek.

V závislosti na prostředí obklopující mobilního robota můžeme metody plánování cesty rozdělit následovně[19]:

a) Plánování cesty se statickými překážkami ve známém prostředí.

b) Plánování cesty se statickými překážkami v neznámem nebo v částečně známém prostředí.

c) Plánování cesty s dynamickými překážkami ve známem prostředí.

d) Plánování cesty s dynamickými překážkami v neznámem nebo v částečně známém prostředí.

Podle modelu obcházení překážek členíme plánování na[1] :

a) Globální plánování

b) Lokální plánování

c) Kombinační plánování

Globální plánování uvažuje přístup, při kterém je vypočítaná kompletní reprezentace konfiguračního prostoru před vlastním hledáním cesty. Tento přístup ale naráží na skutečnost, že vypočítání kompletního konfiguračního prostoru je velmi časové náročné, protože složitost problému roste exponeciálně při zvyšujicím se počtu stupňů volnosti robota. Navíc se dnes většinou užívají plánovače off-line – jsou aktivovány už při existujícím modelu prostředí, vytvoří se cesta která je směřována k řídícím prvkům robota a ten je po krocích provádí. Čas potřebný pro toto plánování není dostatečné malý k tomu, aby umožnil robotu pohyb v dynamickém prostředí.

Lokální plánování vyžaduje pouze omezenou znalost prostředí konfiguračního prostoru. Rozhodnutí o pohybu robota a určení nejlepšího řešení (tedy směru kam se bude robot hýbat) je složeno z lokálního kritéria a heuristiky. Proto je toto řešení rychlejší. Avšak u těchto metod se může stát, že není nalezena cesta (i když ve skutečnosti existuje). Lokální plánování uvažuje plánování jako optimalizační problém, kde nalezení cesty souvisí s optimalizací nějaké dané funkce. Nevýhodou tohoto přístupu je možnost uvíznutí v nějakém lokálním optimu, ze kterého je cíl nedosažitelný nebo velmi obtížně dosažitelný.

Kombinační plánování plánuje cestu kontinuálním konfiguračním prostorem bez potřeby aproximací. Pro tuto vlastnost bývají někdy nazývany přesnými algoritmy.

2.1.1 Základní složky plánování

Podle [2] rozlišujeme tyto složky plánování:

- *Stav* – Při plánování uvažujeme prostor stavů, který zahrnuje všechny možné situace jež mohou nastat. Stav může například reprezentovat polohu robota nebo polohu a

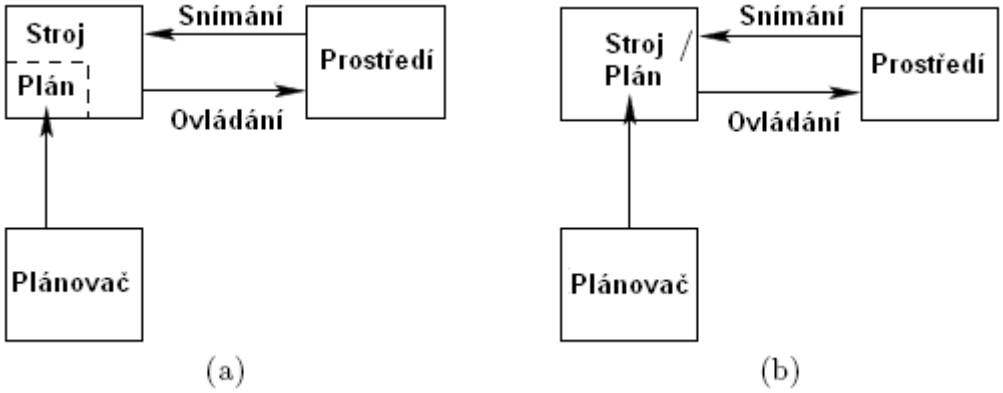
výšku letadla. Prostory stavů, které mohou být reprezentovány jsou buď diskrétní (konečné nebo spočetně nekonečné) nebo kontinuální (nespočetně nekonečné). Definice prostoru stavů je důležitá komponenta při formulaci plánovacího problému a při analyzování a vytváření algoritmu, který jej bude řešit.

- *Čas* – Plánovací problémy zahrnují sekvence řešení, které jsou aplikovány v čase. Čas může být modelován explicitně, jako např. při plánování cesty automobilu, který musí projíždět prostorem plném překážek a to tak, aby bylo vybráno nejrychlejší řešení průjezdu. Druhá možnost je modelovat čas implicitně, tedy v poznatku, že akce musí skončit úspěchem jako např. při řešení Rubikovy kostky. Jiný příklad implicitního chápání času je problém pohybu piana. Řešení pohybu je rozděleno do animací v čase, rychlost pohybu není v plánu zahrnuta. Jako v případě stavů, čas můžeme chápat také jako diskrétní nebo kontinuální.
- *Akce* – Plán vytváří akce které manipulují se stavem. Ve formulaci problému plánování musí být specifikováno, jak se změní stav, když jsou na něj aplikovány akce. Může to být vyjádřeno jako funkce hodnot stavu pro dikrétní čas, nebo jako obyčejnou diferenciální rovnici pro spojitý čas.
- *Startovní a cílový stav* – Plánování začíná běžně ve startovním stavu a končí v cílovém, nebo v jednom ze souboru cílových stavů.
- *Kritérium* – Kritériem stanovujeme požadovaný výstup plánování. Existují dva hlavní směry založené na typu kritéria:
  a) Proveditelnost – najít takový plán, který dosáhne cíle v závislosti na jeho efektivitě.
  b) Optimálnost – najít proveditelný plán, který optimalizuje výkon v nějakém úzce vymezeném způsobu a tímto postupem dosáhne cíle.
- *Plán* – Plán zahrnuje určitou strategii nebo chování rozhodovacího činitele. Plán jednoduše specifikuje posloupnost akcí, ale jen v některých případech. Jestliže je totiž nemožné predikovat budoucí stavy, pak plán specifikuje akce jako funkce stavů. Poté je definována vhodná akce. Tento postup se nazývá postup vzad nebo reaktivní plán.
- *Plánovač cesty* - Plánovač vytváří plán. Může to být stroj nebo člověk. Pokud plán plánuje stroj, jedná se většinou o plánovací algoritmus.

2.1.2 Způsoby využití plánů

- *Provedení plánu*: Plán se provede buď v simulačním prostředí nebo ho vykoná robot spojený s fyzickým prostředím. Jsou dva druhy provedení plánu strojem: Prvním druhem je situace, při které je vyprodukován plánovačem plán který je dekódován a je užit jako vstup do stroje. V tomto případě je stroj definován jako programovatelný, může akceptovat možné plány před provedením. Pokud je jednou plán zadán, je proveden strojem, který se od této chvíle stává nezávislým na plánovači. Druhým druhem je takový plán, který je zakódováním celého stroje. Plán je specializovaný stroj vytvořený pro řešení úkolu zadaného plánovačem. V tomto smyslu může být plán minimalistický a vytvářet nejjednodušší stroj schopný řešit zadanou úlohu.
- *Zdokonalování:* Při zdokonalování se plán určený ke zlepšení využívá jako vstup nového dokonalejšího plánu. Nový plán může pracovat s více problémovými situacemi a může být tedy efektivnější než vstupní. Zdokonalování může být provedeno opakovaně a vytvořit tak řetězec zlepšujících se plánů dokud není vytvořen plán konečný.
- *Hiearchické začlenění:* Hiearchické začlenění umožňuje, aby plán byl součástí akce nějakého většího plánu, je tedy jeho podprogramem. Hiearchické začlenění vytváří kořenový strom plánů. Toto uzpůsobení vede k hiearchickému plánování. Každý vrchol ve stromu je plán. Kořenový vrchol reprezentuje hlavní plán. Potomci každého

vrcholu jsou plány, které jsou začleněny jako akce plánu rodičovského vrcholu. Hloubka stromu nebo počet potomků není limitována.



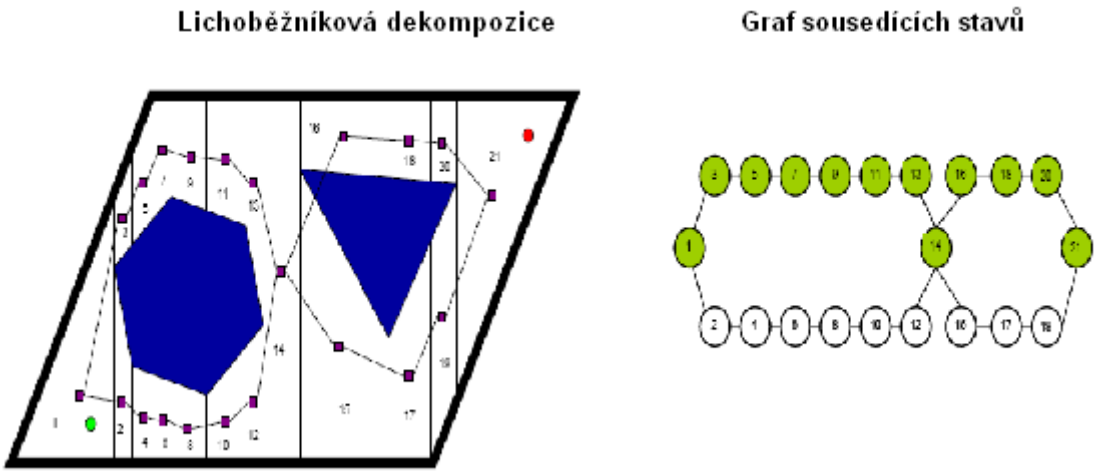
Obr. 1 Dva *způsoby provedení plánu (podle[2]).*

2.2 Metody založené na dekompozici buněk

2.2.1 Přesná dekompozice buněk

Princip této metody zpočívá v rozdělení konfiguračního prostoru do nepřekrývajících se regionů. Poté je vytvořen graf obsahující hranice sousednosti těchto regionů, tento graf je prohledán a výstupem je posloupnost jednotlivých hranic regionů postupujících od startu do cíle nazývaných cesta. Buňky podléhající dekompozici musí splňovat tyto kritéria:

1. geometrie každé bunky by měla bý to nejjednodušší, aby umožnila nejlehčí nalezení řešení
2. malá obtížnost testování sousednosti a nalezení cesty procházející sdílenou hranicí mezi těmito buňkami.



Obr. 2 Lichoběžníková dekompozice. Zdroj:[23]

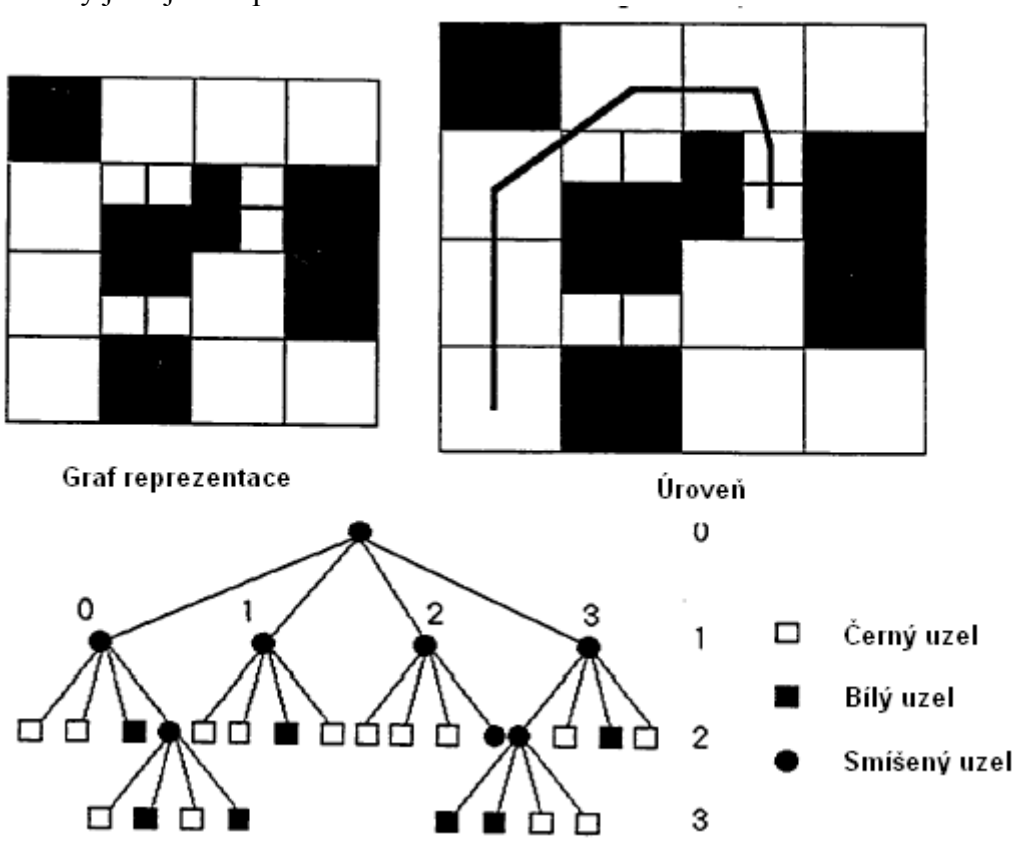
Lichoběžníková dekompozice: Tato metoda přesné dekompozice zpočívá v rozdělení pole do lichoběžníků nebo trojúhelníků. Vytváří se graf sousednosti reprezentující vztahy mezi buňkami. Tato dekompozice vzniká tak, že jednotlivé vrcholy jsou vertikálně obousměrně protahovány dokud nenarazí na hranici pole nebo na překážku. Buňky jsou sousední pokud

spolu sdílí alespoň malý úsek o nenulové délce. Do středu každé lichobežníkové buňky je pak umístěn bod, jehož pomocí propojujeme sousední buňky a je nalezena cesta.

2.2.2 Metody přibližné dekompozice buňek

Tyto metody se odlišují od přesné dekompozice tím, že využívají jednoduchých předdefinovaných zobrazení regionů (například čtverců). Taková reprezentace nedává přesnou prodobu optimální cesty, ale pouze její odhad. Z tohoto důvodu tedy jejich název. Tyto metody využívají stejného principu prohledávání grafu tvořeného sousedícími buňkami. Důvodem ke zjednodušení podoby buněk byla snaha o dekompozici prostoru za použití stále se opakujcícího jednoduchého algoritmu a necitlivost k numericky odhadovaným výpočtům. Důvodem je také to, že jsou značně jednodušší v implementaci než přesné dekompozice. Jedním z metod přibližné dekompozice jsou kvadrantové stromy:

Kvadrantové stromy (Quadtrees): Tato technika dekompozice rozděluje pole do 4 kvadrantů. Každý kvadrant může být buď plný, smíšený nebo prázdný. Kvadrant je plný, pokud je zcela zaplněn objektem nebo jeho částí. Smíšený kvadrant obsahuje pouze část tohoto objektu a není jím zcela vyplněn. Smíšený kvadrant může být během výpočtu rozdělen do dalších 4 podkvadrantů. Tyto podkvadranty mohou být dál a dál rozdělovány do dalších podkvadrantů. Algoritmus vytváří stromovou strukturu, ve které je kvadrant jako kořen a podkvadranty jsou jeho 4 potomci atd.



Obr. 3 Kvadrantová reprezentace a kódování.(Podle [3])

Optimální trasa prokládá pouze prázdné kvadranty. Grafový prohledávací algoritmus prohledává kvadrantové stromy k nalezení souboru na sebe navazujících volných kvadrantů k nalezení trasy ze startu do cíle. Metody užívané k prohledávání jsou například Prohledávání

do hloubky.

2.3 Heuristické metody plánování

Heuristické metody plánování se podle [4] s úspěchem užívají tam, kde zadaný problém má jen málo dimenzí, tj. například když robot má pouze několik stupňů volnosti. Metody plánování pomocí heuristiky můžeme rozdělit do tří skupin: plánování ve statickém a známém prostředí, v částečně známém a dynamickém prostředí.

2.3.1 Základní algoritmy

Plánování cesty můžeme vidět jako problém prohledávání grafu. Bylo vytvořeno množství algoritmů na výpočet nejkratší cesty váženého grafu, mezi nimi jsou nejznámější *Dijkstra* (1959) a A* (1968). Oba algoritmy vypočítávají optimální cestu a mohou být uvažovány jako speciální forma dynamického programování. A* algortimus je velmi podobný Dijkstrově algoritmu s výjimkou toho, že uspoří spoustu výpočetního času soustředěním se na prohledávání stavů nejnadějnějších.

Algoritmus A*

Výpočet_nejkratší_trasy()
1. dokud $(argmin_{s \in OPEN}(g(s)+h(s, s_{goal})) \neq s_{goal})$
2. vyjmi stav s z vrcholu fronty *OPEN*
3. pro všechna $s' \in Následníci(s)$
4. jestliže $(g(s')\{g(s)+c(s,s'))$
5. $g(s')=g(s)+c(s,s')$
6. vlož s' do *OPEN* s hodnotou $(g(s')+h(s', s_{cíl}))$

Hlavní_část_algoritmu()
8. pro všechna $s \in S$
9. $g(s)=\infty$
10. $g(s_{start})=0$
11. $OPEN=\oslash$
12. vlož s_{start} do *OPEN* s hodnotou $(g(s_{start})+h(s_{start}, s_{cíl}))$
13. výpočet_nejkratší_trasy()

Algoritmus pracuje s hodnotou g_s, což je cena cesty z počátečního stavu k dalším stavům s. Na počátku je hodnota g_s nekonečná pro všechny stavy stavového prostoru S. Algoritmus začíná tak, že aktualizuje cenu startovního stavu na nulu a vloží ho do prioritní fronty *OPEN*. Každý stav v této frontě je řazen pomocí klíče, jež je součtem aktuální ceny g_s a heuristického odhadu délky trasy z aktuálního stavu do cíle $h(s, s_{cíl})$. Tato heuristika umožňuje ovlivňovat cenu stavu a je určena k zaměřování cíle. Algoritmus poté vyzvedne stav na vrcholu fronty a aktualizuje ceny okolních stavů výše zmíněným součtem. Startovní stav je vložen do další fronty *CLOSED*. Ostatní nyní již aktualizované stavy jsou vloženy do fronty *OPEN*. Pokud je již ve frontě stav se stejnou polohou a má větší cenu než nově aktualizovaný stav, je stav ve frontě aktualizován, pokud ne, je ponechán existující. Algoritmus pokračuje do chvíle, dokud není z *OPEN* vyzvedáván cílový stav. Jako

modifikace tohoto algoritmu se také používá zpětný A*.

2.3.2 Přírůstkové replánující algoritmy

Tyto algoritmy se využívají pro aplikace v částečně známém prostředí. Příkladem může být robot opatřený senzorem, jehož informace používá postupně při znovuplánování během pohybu v prostředí. Jedním ze z přístupů je novou cestu naplánovat od začátku, tj. z aktuální pozice robota k cíli.K tomuto učelu můžeme využít algoritmus A* popsaný výše. Toto řešení je ale velmi časově náročné (při změně překážky v prostoru, která ani nemá vliv na aktuální optimální cestu, se znovu plánuje nová cesta). Proto se spíše využívá cesta, která využije dosavadní naplánované cesty a upraví je na základě aktuální situace. Příkladem takových algoritmů nacházející své využití v indoor i v outdoor aplikacích jsou algoritmy *D** (*Focussed dynamic D**) a *D*-Lite*.

Základní verze algoritmu D Lite*

Klíč(s)
1. návrat s hodnotou $|min(g(s), rhs(s)) + h(s_{start}, s); min(g(s), rhs(s))|$

Aktualizace_stavu(s)
2. jestliže (s) nebyl předtím navštíven
3. $g(s) = \infty$
4. jestliže($(s \neq s_{cíl})\, rhs(s) = min_{s' \in následník(s)}(c(s, s') + g(s'))$)
5. jestliže $(s \in OPEN)$ vyjmi (s) z *OPEN*
6. jestliže $(g(s) \neq rhs(s))$ vlož (s) do *OPEN* s klíčem (s)

Výpočet_nejkratší_cesty
7. dokud $(min_{s \in OPEN}\, klíč(s)) < klíč(s_{start})\, NEBO\, rhs(s_{start}) \neq g(s_{start})$
8. vyjmi stav (s) s minimálním klíčem z *OPEN*
9. jestliže $(g(s) > rhs(s))$
10. $g(s) = rhs(s)$
11. pro všechna $s' \in Předchůdce(s)$ aktualizuj stav (s')
12. jinak
13. $g(s) = \infty$
14. pro všechna $s' \in Předchůdce(s) \cup (s)$ aktualizuj stav (s')

Hlavní_část()
15. $g(start) = rhs(s_{start}) = \infty;\ g(s_{cíl}) = \infty$
16. $rhs(s_{cíl}) = 0;\ OPEN = \oslash$
17. vlož $s_{cíl}$ do *OPEN* s klíčem($s_{cíl}$)
18. opakuj
19. výpočet_nejkratší_cesty
20. čekej na změny v cenách hran.
21. pro všechny orientované hrany (u, v) se změněnými cenami
22. aktualizuj ceny hran c (u, v)
23. aktualizuj stav (u)

Tyto dva velmi podobné algoritmy jsou schopny se vyrovnat změnám grafu užitém pro plánování. Jednodušší z těchto algoritmů je *D*-Lite*. Tento algoritmus zkonstruuje na

začátku cestu stejným způsobem jako zpětný A*. Pokud v mapě nastane změna (např. cena na hraně se pozmění), pak se aktualizují ceny stavů, jejichž cesty do cíle jsou tímto dotčeny a jsou vloženy do plánovací fronty (*OPEN* seznamu) k šíření efektů těchto změn na zbytek stavového prostoru. Tímto způsobem je ovlivňována jen malá část stavového prostoru. Navíc algoritmus využívá heuristiku k dalšímu omezování se pouze na ty stavy, u nichž předpokládá, že budou mít vliv na cenu cesty. Algoritmus se tedy snaží nalézt nejmenší cenu cesty ze startovního do cílového stavu v konečném stavovém prostoru. Pro vytvoření cesty si ukládá odhad ceny trasy g_s z každého stavu do cíle. Dále pracuje také s $rhs(s)$ což je jednokrokový odhad budoucí ceny za podmínky:

$rhs(s)=0$ *jestliže* $s=s_{cíl}$, *jinak* $rhs(s)=min_{s'\in následníka(s)}(c(s,s')+g(s'))$

kde $následník(s)\in S$ značí soubor následníků stavu S a $c(s,s')$ označuje cenu přesunu ze stavu s do stavu s' (tzv. hraniční cenu). Jako algoritmus A* využívá *D*-Lite* heuristiku a prioritní k frontu k optimální volbě ceny každého stavu. Algoritmus prohledává ve stavovém prostoru zpětně a opakovaně se zaměřuje na startovní stav místo startu cílového. Tím se chce vyrovnat s případnou změnou polohy startovního stavu.

Algoritmus prokázal svojí výhodnost a to hlavně za situací, kdy ke změně stavu dochází v blízkosti robota. Naopak jeho výkon je horší v situaci, kdy v blízkosti robota ke změně stavu nedochází. Jeho výkon klesá a v nkterých případeh je dokonce jeho efektivita horší než u algoritmu A*. Při navigaci v kompletně neznámém prostředí je efektivnější plánovat směrem od startu do cíle než naopak. Je to proto, že optimisticky navrhujeme cenu u stavů které neznáme. Výsledkem je, že stavy které již byly prozkoumány mají cenu vyšší než ceny neprozkoumané a algoritmus, který postupuje vpřed jde pomocí stavů s nižší cenou rychleji k cíli. Oproti tomu při zpětném postupu sice algoritmus postupuje rychle ke zkoumanému prostoru a při jeho dosažení dosáhne stavy s vyšší cenou. Poté začne expandovat velké množství neprozkoumaných stavů k zajištění lepší ceny. Proto je lepší využívat přímého algoritmu A* než algoritmu zpětného A*, když se replánuje od začátku.

2.3.3 Anytime algoritmy

Jestliže je vyžadována rychlá reakce robota na danou situaci a robot musí jednat rychle, nejsou výše zmiňované algoritmy dostatečné z důvodů velkého množství dat, která jsou nutná k řešení takovéhoto problému. V takové situaci musíme mít okamžitě k dispozici nejlepší možné řešení, nejvýhodnější trasu. Anytime algoritmy jsou takové algoritmy u nichž se kvalita výsledku zlepšuje s postupujícím časem. Speciální u anytime algoritmů je jejich schopnost užití dobře definovaných měření kvality ke sledování pokroku v řešení problému a efektivním rozdělování zdrojů.

Jejich kvalita se postupně zlepšuje s postupujícím časem výpočtu. Kvalita výpočtů je ohodnocena tzv. výkonnostním profilem popisujícím závislost kvality výsledku na době výpočtu.V plánování jsou anytime algoritmy použitelná třída deterministických algoritmů. Tyto algoritmy vytvářejí počáteční řešení velmi rychle a toto řešení se dále v čase neustále vylepšuje. Pokud je použitá heuristika konsistentní, tedy pokud pro všechna $s\in S, h(s,s_{cíl})<c(s,s')+h(s',s_{cíl})$ pro všechny následníky $s'\in s, h(s_{cíl},s_{cíl})=0$, pak platí, že po jejím vynásobení inflačním faktorem $\varepsilon>1$ vychází cena, která není větší než ε násobek ceny optimálního řešení. Tato vlastnost byla využita k vyvinutí anytime algoritmů využívajících s úspěchem vážené A* prohledávání, každé se snižujícím se inflačním faktorem, kde každé prohledávání znovuvyužívá pokrok z předcházejícího prohledávání. Více o problematice v[5].

Jedním z těchto algoritmů je anytime opravující se A* *(AR*A*)*. Tento algoritmus se omezuje během zpracovávání pouze na ty stavy, u kterých je cena předchozího hledání rozdílná proti nové hodnotě. Začíná se provedením A* prohledávání s inflačním faktorem

ε_0 , během prohledávání se nicméně expanduje každý stav pouze jednou. Když je stav expandován během částečného prohledávání, pokud je nekonsistentní, tj. jestliže $g_s \neq rhs(s)$ nastávající při změně ceny sousedního stavu, potom není tento stav znovu vložen do fronty stavů k expanzi. Místo toho je vložen do takzvaného *INCONS* seznamu, obsahujícího všechny expandované nekonsistentní stavy. Když je hlavní prohledávání u konce, stavy z *INCONS* seznamu jsou přesunuty do čerstvé prioritní fronty s novými prioritami založenými na novém ε inflačním faktoru a tato fronta je využita pro další prohledávání.

ARA algoritmus*

Klíč(s)

1. návrat s hodnotou $g(s)+\epsilon * h(s_{start}, s)$

Zlepšuj_trasu()

2. dokud $(min_{s\ element OPEN}(klíč(s)) < klíč(s_{start}))$
3. vyjmi s s nejmenším klíčem(s) z *OPEN*
4. $CLOSED = CLOSED \cup \{s\}$
5. pro všechna $s' \in Předchůdci(s)$
6. jestliže s' nebyl navštíven předtím
7. $g(s') = \infty$
8. jestliže $g(s') > c(s', s) + g(s)$
9. $g(s') = c(s'.s) + g(d)$
10. jestliže $s' \notin CLOSED$
11. vlož s' do *OPEN* s klíčem(s')
12. jinak
13. vlož s' do *INCONS*

Hlavní_část()

14. $g(s_{start}) = \infty ; g(s_{cíl}) = 0$
15. $\epsilon = \epsilon_0$
16. $OPEN = CLOSED = INCONS = \varnothing ;$
17. vlož $s_{cíl}$ do *OPEN* s klíčem $(s_{cíl})$
18. vypočítej_zlepšenou_cestu()
19. vyvpiš aktuální ϵ suboptimální řešení
20. dokud $\epsilon > 1$
21. sniž ϵ
22. přesuň stavy z *INCONS* do *OPEN*
23. aktualizuj priority pro všechny $\epsilon \in OPEN$ v závislosti na klíči (s)
24. $CLOSED = \varnothing$
25. vypočítej_zlepšenou_cestu()
26. vypiš aktuální ϵ suboptimální řešení

Tímto způsobem je vylepšována efektivita prohledávání dvěma možnými cestami. Zaprvé je dosaženo velmi rychlého řešení expandováním každého stavu pouze jednou. A zadruhé pouhým znovu prozkoumáním nekonsistentních stavů z minulého vyhledávání je umožněno znovuvyužití výsledků mnoha předchozích prohledávání. Redukováním inflačního faktoru po každém úspěšném prohledávání je dosaženo minimalizace potřebných výpočtů k dosažení řešení.

ARA* je velmi použitelný v doménách s velkým prostorem stavů a suboptimální řešení může být generováno efektivně. Byl úspěšně aplikován na kinematické ruce robota obsahující 20 článků. Je také úspěšně aplikován k výtváření hladkých trajektorií ve vnějším známém prostředí. Tento algoritmus je použitelný pouze na ty případy kdy je požadováno anytime řešení. Dále je *AR*A* použitelný ve statických plánovacích doménách. Pokud překážka v poli změní svoji polohu, plán vytvořený *AR*A* je nepoužitelný je nutné plánovat znovu od začátku. Tato nevýhoda využitelnosti dala vzniknout další třídě algoritmů využitelných v dynamickém prostředí: anytime replánujících algoritmů.

2.3.4 Anytime replánující algoritmy

Výše zmíněné algoritmy se využívají pro případ nejzajímavějších problémů reálného světa. Tyto problémy jsou dynamické (vyžadující replánování) a zároveň jsou komplexní (využívající anytime přístup). Kombinací algoritmů D* a ARA* vznikl algoritmus *AD**(1995). Tento algoritmus využívá sérii prohledávání s následným snižováním inflační ceny tak jako u algoritmu ARA*. Pokud změny prostředí způsobí změny cen hran grafu, jsou tímto ovlivněné stavy vloženy do *OPEN* fronty ke zpracování tak jako u algoritmu *D**. Stavy v této frontě jsou zpracovávány tak dlouho, dokud není nalezeno řešení, které je ε *suboptimální* .

Hlavní funkce AD algoritmu*

1. $g(s_{start}) = rhs(s_{start}) = \infty ; g(s_{cíl}) = \infty$
2. $rhs(s_{cíl}) = 0 ; \epsilon = \epsilon_0 ;$
3. $OPEN = CLOSED = INCONS = \varnothing ;$
4. vlož $s_{cíl}$ do *OPEN* s klíčem $(s_{cíl})$
5. spočti_nebo_zlepši_cestu();
6. vypiš aktuální ϵ suboptimální řešení
7. opakuj
8. jestliže jsou detekovány změny v cenách na hranách aktuální cesty
9. pro všechny orientované hrany (u, v) se změněnými cenami
10. aktualizuj ceny hran c (u, v)
11. aktualizuj stavu (u)
12. jestliže byly pozorovány významné změny cen hran
12. zvyš ϵ nebo přeplánuj od začátku
13. nebo jestliže $\epsilon > 1$
14. sniž ϵ
15. přesuň stavy z I*NCONS* do *OPEN*
16. aktualizuj priority pro všechny $\epsilon \in OPEN$ v závislosti na klíči (s)
17. $CLOSED = \varnothing$
18. vypočítej_zlepšenou_cestu()
19. vypiš aktuální ϵ suboptimální řešení
20. jestliže $\epsilon = 1$
21. čekej na změny na hranách optimální cesty

Algoritmus začíná nastavením inflačního faktoru ε na dostatečně vysokou hodnotu ε_0 , takže počáteční částečně optimální plán může být velmi rychle vygenerován. Poté, pokud není detekována změna ceny na hraně naplánované cesty je snižována hodnota ε . Řešení je vylepšováno do doby, dokud není garantována jeho optimalita, tedy dokud $\varepsilon = 1$.

Tato fáze je stejná jako u algoritmu ARA*. Hodnota ε je snižována, všechny nekonsistentní stavy jsou převáděny z *INCONS* do *OPEN* a *CLOSED*. Pokud je detekována změna cen hran, potom se může stát, že aktuální řešení není ε *suboptimální* . Pokud jsou změny trvalé, může být velmi časově náročné opravit aktuální řešení tak, aby bylo opět ε *suboptimální* . Proto algoritmus zvýší ε tak, aby bylo rychle nalezeno méně optimální řešení. Na rozdíl od samotného algoritmu ARA*, si AD* dokáže poradit se zvýšenou cenou hraničního stavu. Stavy jsou vloženy do *OPEN* listu s klíčovou hodnotou odrážející jejich minulou minimální cenu a aktuální cenu. AD* dokáže reagovat na změny cen hran a zároveň ovlivňovat inflační faktor ε .

AD* je využitelný především v dynamickém prostředí s mnoha stupni volnosti.Využívá se při plánování cesty ve venkovních aplikacích. AD* je v současnosti využívám pro dvě robotické platformy: ARTV a Segway Robotic Mobility Platform.

2.4 Hiearchické plánování cesty

Toto plánování vychází podle [3] z myšlenky rozdělit mapu prostoru do několika na sebe navazujících vrstev.Každá vrstva reprezentuje částečný podprostor v pracovním prostředí. Bylo vytvořeno na základě pozorování lidského plánování akcí. Samotný plánovač je propojen s každou s těchto vrstev. Tato metoda má výhodu ve schopnosti pracovat ve velmi složitém prostředí za využití malého výpočetního času. K nejznámnějším případům tohoto druhu plánovačů patří systémy aplikované v CMU Navlab a Autonomním pozemním vozidle (ALV).Tyto systémy rozdělují podle problém do 4 vrstev: plánovač mise, navigátor, pilot a kontrolér.

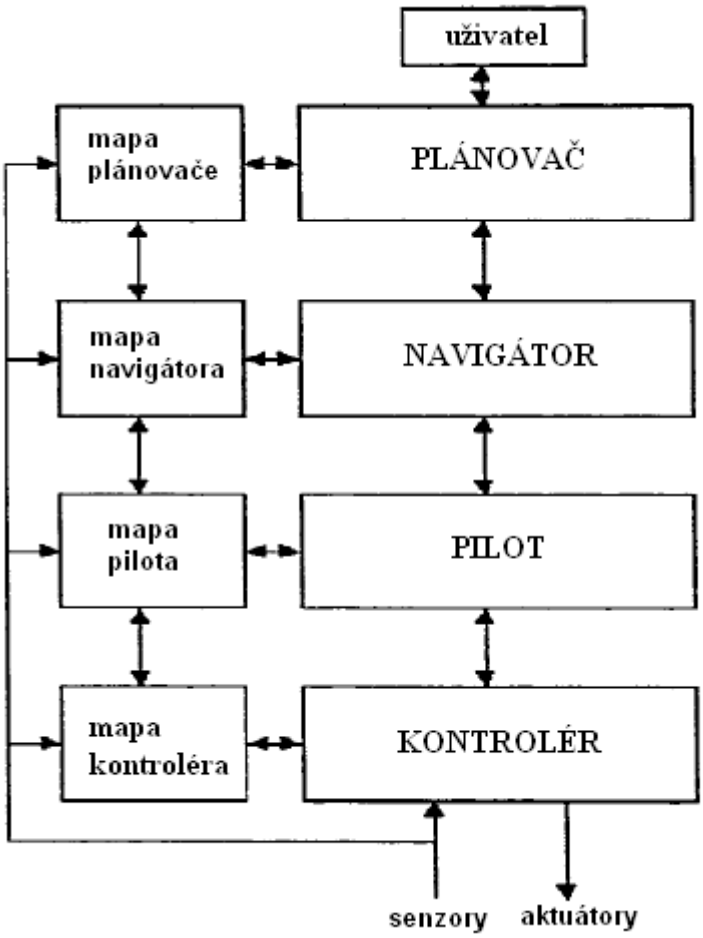
Plánovač mise rozděluje cíl mise (formulovaný operátorem) do série globálních cílů pohybu. Plánovač mise určí mimo bodů cíle také další konstanty jako jsou čas, rychlost atd. Podobně jako při uvažování člověka jako např. při jednoduché úloze vyjet automobilem z garáže, si tento úkol rozdělí do několika na sebe navazujících kroků (1. přijít z místa A do místa B - dveře garáže, 2. otevřít garáž, 3. přejít k autu, 4. otevřít dveře u auta, ...atd.).

Navigátor obdrží tyto cíle a snaží se o nalezení cesty do takto stanovených cílů. Cesta může v případě navigace uvnitř budovy obsahovat trasu ze současného prostoru (místnosti), ve které se nachází robot a mířit do jiných místností. Ve venkovním prostoru může mít naplánovaná trasa délku několik km. Navigátor je závislý na mapě popisující prostředí. Jeho práce je ovlivňována korekcemi od Pilota.

Pilot je se pokouší navigovat senzorů, které snímají okolí robota a vyhýbat se překážkám. I když má Navigátor mapu k plánování trasy, Pilot má tyto informace podrobnější a navíc může reagovat na nepředvídatelné změny prostředí. Pilot sám pomocí senzorů vytváří lokální cestu kolem překážek. Pokud nemá informace ze senzorů a nemůže navigovat (tzn.., že v okolí robota nejsou překážky), vyšle zprávu Navigátorovi o svém selhání. Pilot tedy může aktualizovat mapu prostředí navigátorovi a naopak.

Kontrolér se za pomocí senzorů snaží co nejpřesněji kopírovat trasu vytyčenou pilotem, tedy tuto trasu vykonává.

Výhodou hiearchického plánování je, jak bylo dříve zmíněno, že tento přístup je schopen vyrovnat se s plánováním cesty i ve velmi složitém prostředí. Jeho výhoda je také vhodnost zpracovávání pomocí více procesorů nebo technik paralelního zpracování ke zlepšení výkonu. Toto souvisí s rozdělením plánovacího problému do vrstev. Jeho nevýhoda je především v tom, že práce s více vrstvami vyžaduje komplexnější software pro zpracování této úlohy.



Obr. 4 Klasická plánovací hiearchie.Podle: [3]

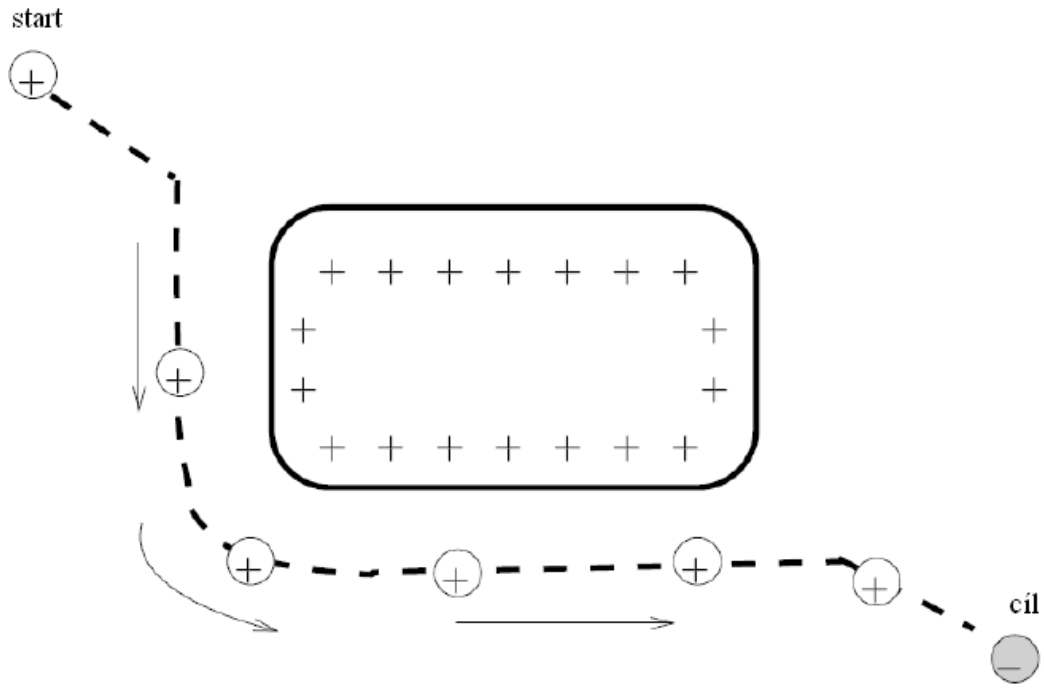
2.5 Umělá potenciálová pole

Při plánování pomocí potenciálových polí(artifical potential fields) se podle [6] stanovuje algoritmus jakési gravitatační pole kolem každé překážky. Toto pole odpuzuje naplánovanou cestu v blízkosti této překážky do volného pole. Naplánovaná cesta je optimální, pokud je nejméně ovlivněna tímto gravitačním polem.

Potenciálová funkce je běžně (ale ne ve všech případech) ve volném prostoru jako suma přitahujícího potenciálu protlačujícího robota k cíli a odpudivého potenciálu odpuzujícího robota při kontaktu s překážkou. Plánování probíhá v opakujícím se cyklu. Při každém opakování je umělá síla $\vec{F}(q) = -\vec{\Delta} U(q)$ ovlivněná potenciálnvou funkcí za aktuální konfigurace pokládána za nejlepší možný směr pohybu a plánovač prochází kolem tohoto směru s malým přídavkem. Metoda byla původně vyvinuta jako on-line přístup k obcházení překážek, při kterém robot nezná přesný tvar překážek a vnímá je až při vlastním pohybu prostorem. Důraz byl kladen na efektivitu real-time než na dosažení cíle. Z tohoto důvodu se může algoritmus zaseknout v nějakém lokálním minimu. Tato nevýhoda je vyvažována použitím nějakého jednoduchého algoritmu na prohledávání grafu, který dokáže algoritmus vytáhnout z tohoto minima. Nicméně i když jsou tyto metody užity, problém s lokálním minimem zůstává dále zdrojem neefektivity tohoto algoritmu. Proto se v zásadě přichází se dvěma přístupy:

1. vytvoření potenciálové funkce s velmi málo nebo žádným lokálním minimem
2. vytvoření prohledávacího algoritmu se schopností efektivně opustit lokální minimum.

Metody založeé na potenciálových polích se mohou stát neefektivní, pokud existuje malé množství nebo jediná optimální cesta prostorem. tj. pokud v prostoru existuje velké množství překážek.



Obr. 5 Základní myšlenka umělých potenciálových polí.Podle [21].

Jednou z technik využívající tuto metodu popsal C.W.Warren ve své práci[7], v níž se nejprve naplánuje libovolná cesta přes konfigurační prostor a poté se upravuje vzhledem k potenciálovým polím. V této metodě se potenciálové pole se skládá ze dvou částí: z potenciálového pole ležícího uvnitř překážky a potenciálového obklopujícího tuto překážku. Pokud cesta prochází vnitřkem potenciálového pole, je tato cesta silně odmrštěna do volného prostoru. Algoritmus musí být takový, aby dokázal odmrštit cestu nezávisle na tom, jakým směrem cesta přes tuto překážku prochází:

$U_{uvnitř} = U_{max} * (1 - (\frac{R_{uvnitř}}{R_{max}})) + U_{offset}$ kde $U_{uvnitř}$ je potenciálové pole uvnitř překážky, U_{max} je maximální potenciálové pole, $R_{uvnitř}$ je vzdálenost z daného bodu do těžiště, U_{max} je maximální rádius z těžiště, U_{offset} je posunutí od základny pro vnitřní potenciál.

Tyto algoritmy jsou výhodné pro prostředí s malým počtem překážek. Pokud počet překážek vzrůstá, velmi se zvyšuje potřebný výpočetní čas. Robot může uvíznout a nenalezne vhodnou cestu.

2.6 Plánování za užití genetických algoritmů

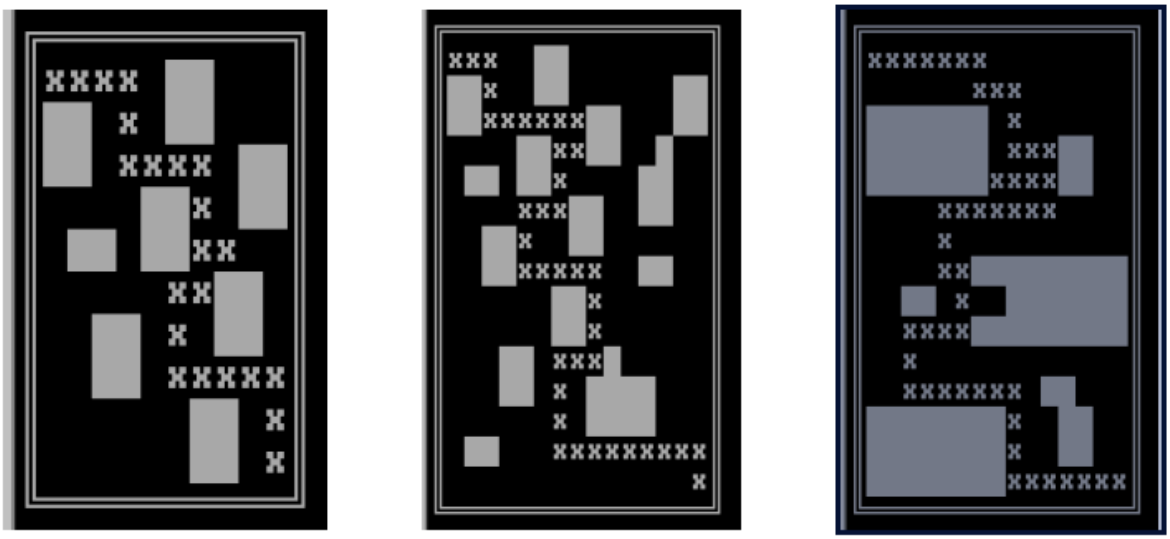
Genetické algoritmy jsou podle adaptivní prohledávací procedury založené na zjednodušených modelech evoluce. Jednoduché GA jsou podle [8] tvořeny ze tří základních prvků: reprodukce, křížení a mutace.

Reprodukce je proces, při kterém jsou jednotlivé řetězce kopírovány v závislosti na hodnotách jejich fitness, což je měřítko vhodnosti řešení, které reprezentuje daný řetezec. Kopírováním řetezců v závislosti na jejich fitness funkci vede k tomu, že vhodnější řetezce mají šanci mít potomstvo v další generaci. Jakmile je řetězec vybrán k reprodukci, je vytvořena jeho přesná kopie. Řetezec je pak vložen do pokusného souboru pro další akce genetického operátoru.

Křížení zpočívá v dvou krocích. Členové pokusného souboru jsou náhodně spárováni a poté je provedeno křížení.

Mutace je v genetickém algoritmu potřebná protože proces křížení a reprodukce vede často ke ztrátě genetického materiálu. V jednoduchých genetických algoritmech je tento proces umožněn náhodným prohozením hodnoty v řetezci.

Genetický algoritmus startuje podle [9] s populací jedinců (chromozómů). Každý jedinec reprezentuje možné řešení pro daný problém. V problému plánování každý jedinec může reprezentovat cestu ze startovního k cílovému bodu. Každý jedinec poté obdrží hodnotu fitness, která reprezentuje, jak který jedinec splňuje kritéria problému. Jedinci vybraní za použití fitness funkce jsou označeni jako rodiče. Rodiče vytvoří pomocí křížení nebo mutace nové potomstvo. Proces pokračuje tak dlouho, dokud není nalezeno optimální řešení nebo řešení blízké optimálnímu.



Obr. 6 Příklad procházení polem za užití genetického algoritmu. Zdroj: [9]

Základní rozlišovací znak genetických algoritmů je to, jak reprezentují prostor řešení pro daný problém. Každý jedinec v prostoru řešení musí reprezentovat platný prvek. Řešení jsou překódovány do genotypů – reprezentačních modelů, které mohou být manipulovány genetickým algoritmem. Fitness funkce je užita ke zkoumání každého genotypu zda odpovídá cílům úkolu. Genotyp musí být schopen reprezentovat správnou cestu, aby byl schopen dostát požadavkům navigace robota, totiž vytvářet nejkratší cestu neznámým prostředím s překážkami. Navíc musí být struktury algoritmu a fitness funkce jednoduché, aby proces mohl probíhat co nejrychleji.

Existuje mnoho moderních metod plánování cesty pomocí genetických algoritmů. Jedním z přístupů je kombinace fuzzy logiky s genetickými algoritmy, jak je popsáno v práci[10]. V tomto přístupu genetický algoritmus reprezentuje fuzzy pravidla, která řídí

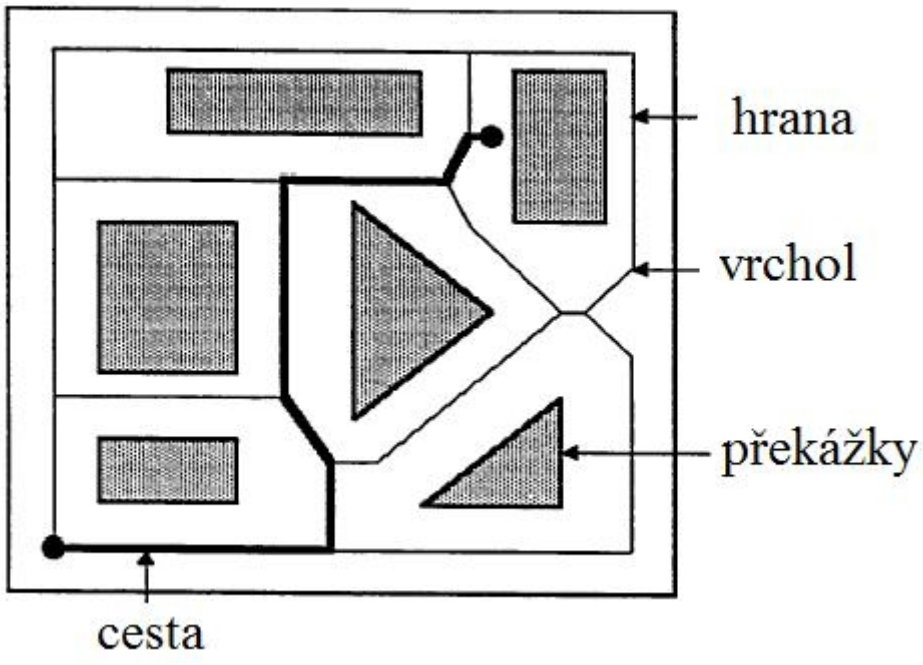
navigaci robota a tak se genetický algoritmus vyvine v nejlepší soubor pravidel. I když tento přístup může vytvořit vhodnou cestu nejistým prostorem, problémem se stává narůstající genetický soubor pravidel, který posléze vyžaduje soubor variant těchto pravidel. Komplexní genetický algoritmus vyžaduje stále více výpočetního času, což ovlivňuje aktuální výkonnost robota pohybujícího se prostředím. Jiným přístupem je užití genetického algoritmu, který reprezentuje místní vzdálenost a směr pohybu namísto reprezentace celé cesty.

2.7 Metody založené na cestovní mapě

Tyto metody jsou podle [6] založeny na myšlence s cílem sestavit topologický graf nazývaný cestovní mapa (roadmap) *R*, který efektivně řeší problém vícenásobného dotazování. Zajistit spojení ve volném prostoru C_{free} ve formě jednodimenzionálních propojení křivkami – cestovní mapu ležící v oblasti C_{free} nebo v uzavřeném (C_{free}) . Když jsou křivky zkonstruovány, cestovní mapa *R* je použita jako soubor standardizovaných cest. Plánování je redukováno na propojení startovního a cílového stavu s mapou *R* a prohledávání cesty v *R*. Prvním základním úkolem těchto metod je konstrukce *R*. Byly vytvořeny metody používající různé přístupy, produkující různé druhy cestovnách map jako jsou například Voroného diagramy, grafy viditelnosti (visibility graphs), expresní sítě(freeway nets0, nebo siluety (silhouettes).

2.7.1 Voroného diagramy

Na začátku plánování je zkonstruován Voroného graf. Prostor můžou tvořit tvořit polygony nebo zakřivené plochy. Hranice diagramu posléze slouží jako liniové segmenty tvořící základ pro prohledávání.Algoritmus se poté snaží pospojovat tyto linie od startu do cíle.



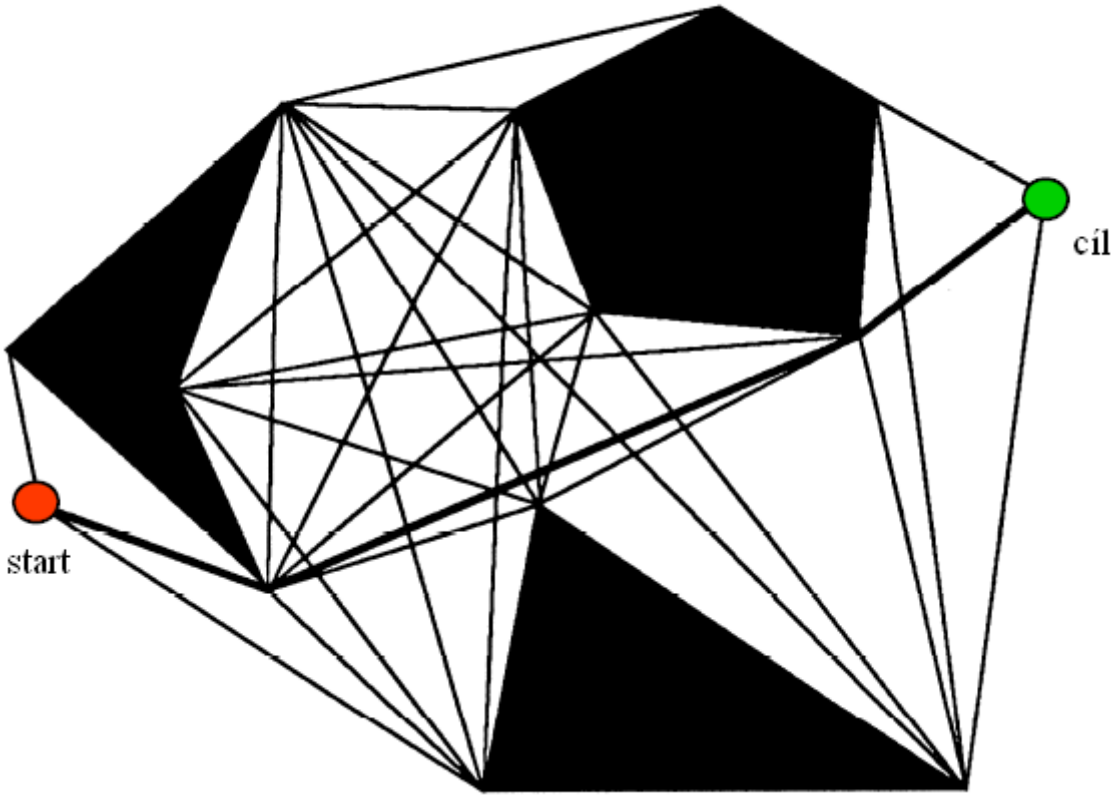
Obr. 7 Voroného graf a generovaná cesta. Podle:[3]

Výhodou tohoto algoritmu je, že počet linií, které musí algoritmus zpracovávat, je

velmi malý a ve známem prostředí jsou cesty vypočítány velmi rychle. Mezi jeho nevýhody patří malá aplikovatelnost v prostředí s pohybujícími se překážkami, protože nejdelší čas zabírá konstrukce Voroného diagramu.

2.7.2 Graf viditelnosti

Tato jednoduchá metoda je jednou z nejstarších plánovačů trasy a byla velmi využívaná pro plánování cesty. Je typicky aplikována v dvoudimensionálních prostorech s polygonálními překážkami. Graf je vytvořen jednoduduchým propojením vrcholů překážek úsečkami. Úsečkami můžeme propojit také vrcholy konfiguračního prostoru se startem a cílem.Usečka nesmí protnout při spojování dvou vrcholů žádnou překážku, odtud název metody graf viditelnosti.



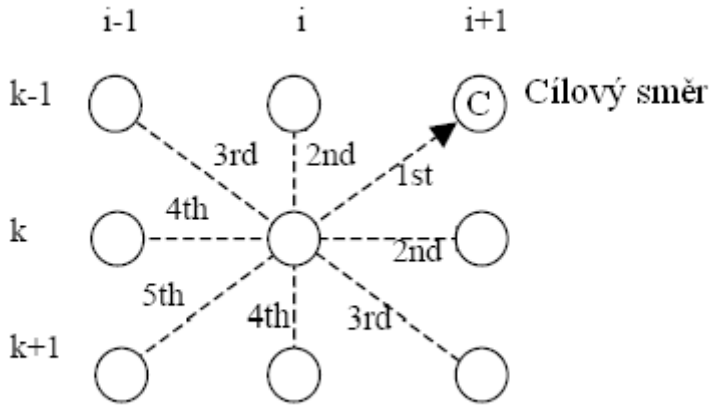
Obr. 8 Graf viditelnosti. Tlustá linie vyznačuje cestu ze startu do cíle. Podle:[22]

2.8 Plánování cesty pomocí neuronových sítí

Neuronové sítě využívají ke své reprezentace model lidského mozku. Jednou z metod je modifikace Hopfieldovy neuronové sítě[11]. Je získán prostor, který neobsahuje překážky a umožňuje měnit aktivační úroveň jednotlivých neuronů ze startovního bodu do cílového. Cesta je konstruována tak, že se hledá nejvýše aktivovaný neuron. Tato metoda využívá asymetrickou váhovou mapu popsanou níže.

Hopfieldova síť je jednovrstvá síť s pevnými vahami. Má periodický charakter. Výstup okolních neuronů v_j je vstupem po zvážení a sečtení pro neuron u_i . V každém

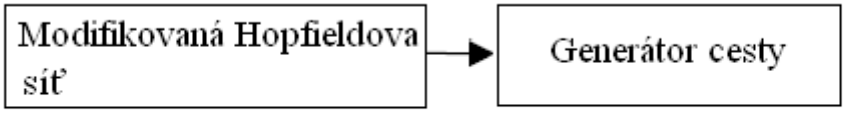
kroku je aktivován a aktualizován pouze jediný náhodně vybraný neuron i . Jako aktivační funkce je v diskrétním prostředí použita znaménková funkce, pro analogovou Hopfieldovu síť se užívá sigmoidní nebo hyperbolická. Konstrukce váhové matice: Konfigurační prostor C je namapován do prostoru neuronů. Každý neuron v rámci své podmnožinny C_i je propojen s množinou konfiguračním prostorem C . Všechny neurony tak mohou být reprezentovány jako konfigurační prostor. Každý neuron má citlivé pole RF_i což je podmnožina jedinců v jeho okolí. Každý jedinec i je s propojen s každým dalším jedincem v RF_i . Žádné další spojení mimo citlivé pole neexistuje. Váhová matice je vytvořena pomocí vzdálenosti mezi cílovým a každým dalším neuronem. Největší váha je nastavena neuronům obklopujícím cílový stav. Váha se snižuje vzdalováním se od cílového stavu. Cesta je generována z nejvýše aktivovaného neuronu k cíli. Systémový model takto popsaný je známý jako Hopfieldova neuronová síť.



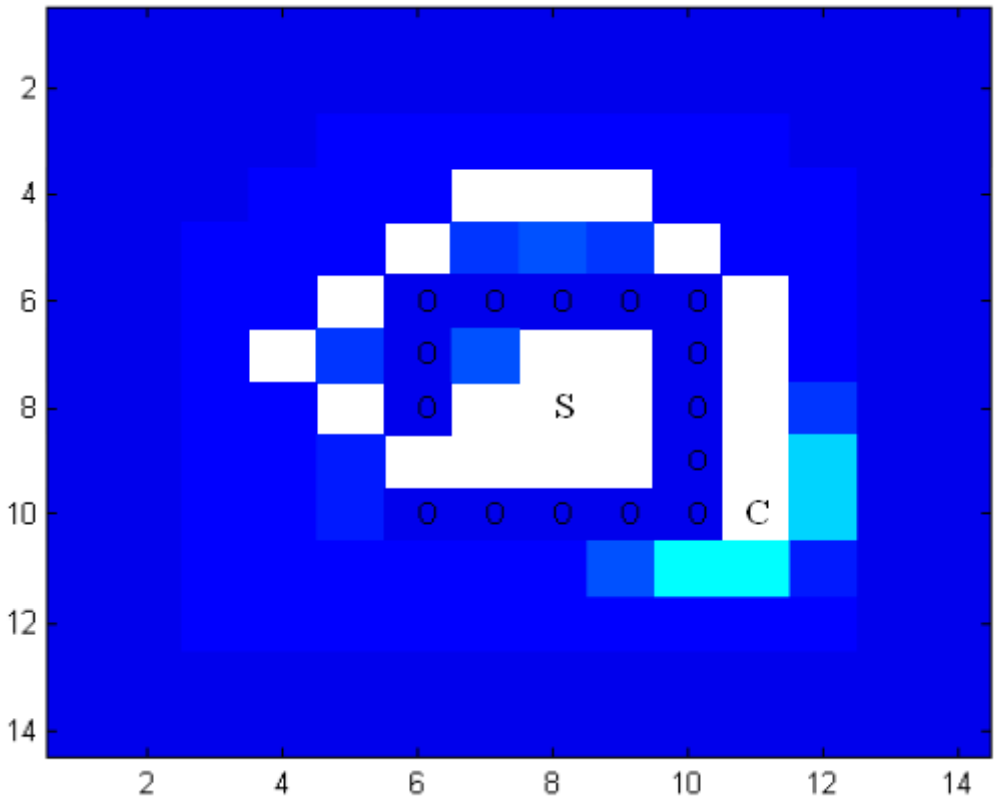
Obr. 9 Asymetrická váhová mapa. Podle:[11]

V modifikované Hopfieldově neuronové síti jsou startovní a cílový bod nastaveny maximálně aktivovaným neuronem.

V prostoru tedy vznikají dva extrémy. V prostoru se od těchto extrémů odvíjí další (nižší) úrovně aktivovaných neuronů. Překážky jsou nastaveny jako neaktivní neurony. Nastavením spojovacích vah zmíněných výše mohou být znatelně odlišeny úrovně neuronů mířící od startovního stavu k cíli. Prohledáním každého nejvyššího aktivního neuronu při porcházení ze startovního stavu může být velmi lehce získána cesta k cíli. Start, cíl a překážky jsou označeny S, C a P. Neurony které jsou nejblíže startovnímu nebo cílovému stavu jsou nevýše aktivovány. Generátor cesty se soustředí při dalším kroku na sousední stavy. Startovní neuron je považován za aktuálního jedince. Prozkoumá okolí a vyhledá dva nejvíce aktivované jedince:Tyto dva jedinci jsou dále zkoumáni, je zjišťováno jak aktiivní jedinci je obklopují. Jedinec s aktivnějším okolím se potom stává aktuálním stavem. Tímto způsobem je konstruována cesta. Tím, že algoritmus nezkoumá stav aktuální je zamezeno vytváření zacyklení.



Obr. 10 Ukázka systémového modelu. Podle [11]



Obr. 11 Příklad cesty vytvořené asymetrickou váhovou maticí.Podle[11]

Další možnost využití neuronové sítě je popsán v práci[12]**.** Tato práce je věnována aplikaci neuronové sítě na odporovou mřížku (resistive grid) pro plánování cesty. Odporová mřížka, nebo také Laplaceova plánovací technika, se vyrovnává s problémem lokálního minima a garantuje nalezení cesty (pokud tato existuje). Neuronová síť se skládá ze dvou vrstev. Ve vrchní vrstvě laterální spojení mezi neurony zhodnocují informace o potenciálech sousedních polí v mřížce. Spodní vrstva reprezentuje prostorovou pamět, ve které jsou uloženy pozice překážek a cíle. Každý neuron ve vrchní vrstvě obdrží jednoduchý vstup od vrstvy spodní, kteréžto poloha souhlasí s polohou vrchní vrstvy. Vstup je určen k hodnocení potenciálů vybraných uzlů využitých v odporové mřížce. Součinnost mezi těmito dvěma vrstvami nám dává flexibilní algoritmus snadno aplikovatelný v nových prostředích. Příklad cesty vytvořené asymetrickou váhovou maticí.Podle[11].

Další z aplikací neuronových sítí na plánování cesty je popsán v práci [13]. Trajektorie v reálném čase je generována dynamicky aktivovanou scénou. Je vytvořena topologiicky orientovaná architektura neuronové sítě, u které dynamicky aktivované, neuronové prostředí reprezentuje dynamicky variabilní prostředí. Vhodně zvolenými externími vstupy z různorodého prostředí a vnitřních neuronových spojení je docíleno, že překážky a cíl představují vrcholy a údolí představuje pole, ve kterém jsou vytvářeny neuronové sítě. Cíl přitahuje globálně robota přes celý prostor, překážky mají vliv pouze jako lokální efekt vyhýbání se překážkám.

3 DYNAMICKÉ PROGRAMOVÁNÍ V PLÁNOVÁNÍ CESTY ROBOTA A JEHO APLIKACE

Dynamické programování je jednou ze základních tříd metod k řešení problému posilovaného učení. Posilované učení je učení mapování situace do akcí tak, aby bylo dosaženo maximální odměny. Agent není tázán jakou akci má provést, nýbrž jaká akce mu po přezkoušení možných řešení přináší nejvyšší odměnu. V nejzajímavějších případech akce mohou obsahovat nejen následnou odměnu, ale také další situace, což může vést k řetezcům následných odměn[14].

Princip optimality a dynamické programování nabízí podle [15] obrovskou výhodu v redukci výpočetního času v mnoha metodách plánování cesty. Dynamické programování je matematický přístup využívaný k řešení optimalizačních úloh. Richard Bellman ho uvedl ve své práci v roce 1957. Je založen na zkoumání víceetapových rozhodovacích procesů, u nichž etapy jsou za sebou postupně seřazeny v čase. Umožňuje např. řešení numerické, ve tvaru analytického vyjádření nebo řešení ve tvaru postupných aproximací. Bývá využíván při řešení optimalizačních úloh deterministických i stochastických v různých oborech – matematice, fyzice, chemii, technice. Jeho vysoká efektivnost je ověřena při řešení některých úloh se silnými omezujícími podmínkami. Mezi jeho nevýhody patří hlubší matematická příprava v kontrastu např. s metodami lineárního programování, jelikož problém nemusí být vždy charakterizován systémem rovnic nebo nerovností, a tedy neexistuje dostatečně obecný algoritmus k řešení některých problémů.

Markovovy řetězce jsou jsou speciálním typem stochastického procesu. Uvažujme systém, který se v každé z diskrétních etap $0,1,2\ldots$ nachází v jednom z konečného počtu N různých stavů. Pravděpodobnost P_{ij} přechodu ze stavu i v etapě n do stavu j v etapě $n+1$ je nezávislá na n i na stavech, kterými soustava prošla v etapách $0,1,2,\ldots,n-1$. Platí

$$P_{ij} \wedge 0, \sum_{j=1}^{N} P_{ij} = 1, i, j = 1,2,\ldots,N.$$

Matice P_{ij} je nazývá maticí přechodu. Přirozeným propojením Markovských řetezců s dynamickým programováním se ustálil pojem Markovův rozhodovací proces.

3.1 Markovské rozhodovací procesy pro plánování cesty

Úloha posilovaného učení která vyhovuje Markovově vlastnosti se nazývá Markovský rozhodovací proces (MRP). Jestliže je prostor stavů a akcí konečný, je nazýván *konečný MRP*.

Markovské rozhodovací procesy jsou všeobecně užívány jako modely pro rozhodování za nejistoty. Markovův rozhodovací proces sekvenčně vytváří rozhodnutí, které obsahují akce transformující stav do jednoho z několika možných následných stavů, přičemž má každý jednotlivý možný následný stav určitou míru pravděpodobnosti přechodu. Řešení vytváří formu mapování ze stavů do akcí nazývanou strategie. Postup je vykonnán zkoumáním aktuálního stavu a vykonáním akce pro něj vhodné. Optimální postup může být nalezen za použití algoritmů dynamického programování jako je iterace strategií (policy iteration) nebo iterace hodnot (value iteration). Nevýhodou těchto metod dynamického programování je, že prohledávají celý prostor stavů. Touto metodou se totiž nalezne strategie postupu pro každý možný startovní stav. V průběhu času byly vyvinuty metoty pro řešení tohoto problému. Některým z nich je věnována tato práce. Optimální strategií je taková strategie, která maximalizuje (nebo naopak minimalizuje) nějakou předem definovanou

účelovou funkci. Metoda optimalizace (metoda identifikující optimální strategii) závisí na formě účelové funkce, tj. na optimalizačním kritériu. Volba vhodného kritéria závisí na povaze prostředí (např. jsou rozdílné při nekonečném nebo konečném horizontu).

V Markovském rozhodovacím procesu uvažujeme množinu M obsahující čtveřici prvků $\{S, A, T, C\}$.

- S – značí konečnou množinu stavů, z nichž některé mohou být terminální (cílové).
- A – značí konečnou množinu akcí, pro každý stav je možno určit určitou množinu akcí.
- T – značí tranzitní (přechodovou) funkci nebo model, ve kterém $T(s_i, a, sr_j)$ je pravděpodobnost přechodu do stavu s_j při aplikaci akce na stav s_i .
- C – značí cenovou funkci , ve které $C(s_i, a)$ určuje očekávanou cenu aplikace akce a ve stavu s_j .

Strategie $\pi : S \rightarrow A$, která mapuje stavy do akcí, specifikuje danou akci v každém stavu světa. Pomocí zadané strategie π a cenové funkce C můžeme definovat hodnotu stavu s_i jako její celkovou cenu za dané strategie π . Hodnota stavu s_i daného strategií π můžeme popsat vzorcem:

$$V_\pi(s_i) = C(s_i, \pi(s_i)) + \sum_{(s_j \in S)} P(s_i, \pi(s_i), s_j) * V_\pi(s_j)$$

Strategie π je optimální, pokud $V_\pi(s_i) \wedge V_\pi{}'(s_i)$ pro všechna $s_i \in S$ a strategie π' .

3.1.1 Výpočet optimální strategie a hodnoty v MRP

Základní součástí dynamického programování respektive posilovaného učení je použití hodnotové funkce za účelem organizace a strukturování hledání optimální strategie[14].

Hodnotová funkce: většina algoritmů založené na posilovaném učení využívají odhadu hodnotové funkce, funkcí stavů nebo párů stav-akce, které odhadují *jak dobré* je pro agenta být v daném stavu (nebo jak dobré je provést danou akci v daném stavu). Pod termínem *jak dobré* v daném případě rozumíme budoucí odměny, které může agent očekávat. Hodnota stavu s za politiky π popsaná jako $V^\pi(s)$ je očekávaný zisk při startu v s za následování strategie π .

Zlepšování strategie: zlepšováním strategie se rozumí takový postup při kterém se vytváří nová strategie, která upravuje tu stávající. Zvážením akcí a *ve stavu* s *můžeme buď následovat existující strategii* π *nebo přejít na novou strategii* π' .

Oceňování strategie: K vytvoření nové aproximace V_{k+1} z hodnoty V_k algoritmus opakuje pro každý stav s daný postup: náhrada staré hodnoty V_k za novou V_{k+1} , která je získána jako minimální stará hodnota následnických stavů stavu s a očekávaných odměn při postupu do těchto stavů. Každé opakování zálohuje aktuální hodnotu každého stavu dokud na něj není provedena další aproximace. Tento proces se nazývá *plná záloha* (full backup)

K výpočtu optimální strategie a optimální ceny stavu se k řešení konečných Markovských rozhodovacích procesů se využívají nejhojněji dvě základní metody: Iterace hodnot a iterace strategií. Klasické metody využívají procesu postupných průchodů (sweeps) přes prostor stavů, které pro každý stav provádějí plnou zálohu. Každá záloha aktualizuje

hodnotu stavu založenou na hodnotě všech možných následnických stavů a pravděpodobnosti jejich dosažení.

- *Iterace strategií:* V této metodě se pracuje se vzorcem daném iterací hodnoty, pak se umožní strategii aktualizovat tak, že v akce provedená v každém stavu poskytuje minimální hodnotu pro nově vypočítané hodnoty sousedů tohoto stavu. Tento proces pokračuje, dokud existuje rozdíl mezi novou strategií a předcházející, tedy $\Pi^{(i+1)} = \Pi^{(i)}$. Proces vychází ze dvou metod: oceňování strategie a zlepšování strategie.

  1. Inicializace
     $V(s) \in R$ a $\pi(s) \in A(s)$ náhodně pro všechna $s \in S$
  2. Oceňování strategie
     opakuj dokud není $\Delta < \Theta$ (theta je nějaké malé kladné číslo)
     $\Delta \leftarrow 0$
     pro každé $s \in S$:
     $v \leftarrow V(s)$
     $V(s) \leftarrow \Sigma_{s'} P_{ss'}^{\pi(s)} [R_{ss'}^{\pi(s)} + \gamma V(s')]$
     $\Delta \leftarrow (\Delta, |v - V(s)|)$
  3. Zlepšování strategie
     strategie-stabilní $\leftarrow$ pravda
     pro každé $s \in S$:
     $b \leftarrow \pi(s)$
     $\pi(s) \leftarrow arg\ max_a \Sigma_{s'} \overset{a}{P}_{ss'} [\overset{a}{R}_{ss'} + \gamma V(s')]$
     jestliže $b \neq \pi(s)$ pak strategie-stabilní $\leftarrow$ nepravda
     jestliže strategie-stabilní potom stop, jinak jdi na krok 2.

- *Iterace hodnot:* Jedna z nevýhod iterace strategií je, že při každém opakování se provádí oceňování strategie, která sama o sobě může být sérií opakování průchodů přes prostor stavů a tím tento proces zdržovat. Pokud je oceňování strategie prováděno opakovaně, potom konvergence do V^{π} nastává pouze v limitě.Ve vetšině případů nemusíme čekad na přesnou konvergenci k optimální hodnotě, stačí nám kratší cyklus. Proto se snažíme zkrátit proces oceňování strategie a to pouze na jediný postup prostorem (jedna záloha pro každý stav). Tento algoritmus se nazývá iterace hodnot. Vstupem této metody je horní hraniční hodnota $V^{(0)}$ a hodnota v i-té itraci se vypočte podle vzorce:
  $$V^{(i+1)}(s_i) = \min_{(a \in A)(s_i \in S)} \Sigma P(s_i, a, s_j) * (C(s_i, s_j) + V^{(i)}(s_j))$$
  pro každý element $s_i \in S$. Pomocí tohoto postupu funkce konverguje k optimální hodnotové funkci. Optimální strategie je extrahována vybíráním akce v každém stavu, který dosáhne minimální hodnoty specifikované hodnotovou funkcí. Algoritmus končí, pokud se hodnotová funkce změní v kroku o pouze velmi malou hodnotu.

1. inicializuj V náhodně, například $V(s)=0$, pro všechna $s \in S^+$
2. opakuj dokud $\Delta < \epsilon$ (epsilon je nějaké malé kladné číslo, označuje *prahovou mez*)
   Pro každé $s \in S$:
   $V(s) \leftarrow max_a \Sigma_{s'} P_{ss'}^{\pi(s)} [R_{ss'}^{\pi(s)} + \gamma V(s')]$
   $\Delta \leftarrow max(\Delta, |v - V(s)|)$
   3.výstup deteministické strategie π , ve které
   $\pi(s) \leftarrow arg\, max_a \Sigma_{s'} \overset{a}{P}_{ss'} [\overset{a}{R}_{ss'} + \gamma V(s')]$

3.1.2 Metody pro efektivnější výpočty MRP

Hodnotová iterace a iterace strategií jsou výborné metody pro optimalizaci. V případě plánování cesty robota se musí ovšem klást větší důraz na některé stavy v prostoru. Je to z toho důvodu , že výše popsané metody uvažují celáý prostor stavů. V problému plánování cesty se ale stačí soustředit jen na úzkou výseč stavů vedoucí od startovního stavu do cíle. Metody umožnující zefektivnění těchto dvou postupů jsou např.

- *Dynamické programování v reálném čase* – algoritmus postupuje tak, že omezuje skupinu zkoumaných stavů na velmi úzký výsek celkového prostoru. Začíná s přípustným odhadem hodnoty pro každý stav světa. Poté aplikuje sérii simulovaných přechodů přes prostředí, která začíná ze startovního stavu a pokračuje aplikací *hladové strategie* vzhledem k aktuálnímu odhadu hodnoty. Hodnoty stavů jsou aktualizovány za použití iterace hodnoty, když jsou navštíveny souborem simulovaných přechodů.

- *Propagace obálky* - je metoda redukce prostoru stavů do „obálky“, která se pak uvažuje v dalším výpočtu[16]. Jako u dynamického programování v reálném čase začíná algoritmus s přípustným odhadem hodnot všech stavů světa. V první fázi aplikuje algoritmus *prohledávání do hloubky* ze startovního do cílového stavu.
  Poté se z takto vytvořené cesty vyloučí nadbytečné stavy a vyberou ty nejkratší. Pak je tato cesta obklopena sousedními stavy a vytvoří obálku. Potom se proces rozdělí do dvou fází: expanze obálky – to je fáze, při které jsou simulovány přechody přes stavy v obálce ze startovního stavu pomocí dané strategie. Pokud se při této expanzi narazí na stav, který není v obálce, je označen jako kandidát do obálky. Poté co jsou provedeny všechny simulované přechody, jsou stavy navštívené nejvíce často vloženy do obálky. Obálka je poté posílena výpočtem cest ze všech těchto stavů zpět do obálky a přidáním sousedů těchto stavů do obálky. Následuje fáze generování strategie, při které je provedeno dynamické programování na stavy v obálce k výpočtu optimální (relativně k obálce) strategie pro každý tento stav. Vhodná metoda je buď hodnotová iterace nebo iterace strategií.

- *LAO** - podobně jako algoritmus propagace obálky i tento využívá dvě fáze: fáze expanze a fázi vytváření strategie. Fáze expanze obhospodařuje na rozdíl od obálkové metody pouze takové stavy, které jsou v daném okamžiku dosažitelné ze startovního stavu. V expanzní části jsou všechny cípové stavy přidány do obálky a v další fázi je vybrán nejlepší stav pro dsalší expanzi.

- *Prioritní přeskakování (Prioritized Sweeping)* – algoritmus popsaný v [20] byl vyvinut k aplikaci posilovaného učení na stochastické Markovovy systémy. Místo provádění množství přechodů v poli spojeném s aktualizací každého jednotlivého stavu v něm (pomocí hodnotové iterace nebo strategie iterace) je obsluhována proritní fronta a aktualizovány stavy podle jejich pořadí v této prioritní frontě. Jsou dva různé způsoby aplikace prioritního skákání na plánování MRP. První způsob je inicializace všech stavů pomocí přípustných heuristických hodnot a poté šíření aktualizace ceny dopředně ze startu do cíle. Druhý přístup je inicializace všech stavů s výjimkou startovního nekonečnými hodnotami a šíření z cílového stavu do startovního. V prvním případě je prioritní fronta nastartována vložením startovního stavu do fronty, ve druhém se do prioritní fronty nasazují stavy obklopující cílový stav. V obou případech má algoritmus stejný postup: stav s nejvyšší prioritou je vyjmut z fronty a jeho hodnota je aktualizována. Rozdíl mezi hodnotou původní a hodnotou nově aktualizovanou tohoto stavu se nazývá delta hodnota Δ . Tento stav je poté vložen do fronty pokud jeho delta hodnota nespadne pod určitou mez. Podobně jako tento stav jsou do fronty umístěny i jeho sousední stavy,které si také počítají svou vlastní delta hodnotu. Pokud existuje ve frontě stejný stav s nižší prioritou než má stav vkládaný, je tento aktualizován tímto stavem. Algoritmus pokračuje dokud není prioritní fronta prázdná.
- Směrovanédynamické programování (Focussed Dynamic Programming) – je metodou využívající výhod prioritního skákání a dalších úspěšných metod pro plánování nejkratší cesty. Jemu a úpravě algoritmu LAO* je věnována tato práce a je aplikováno srovnání těchto algoritmů.

3.2 Algoritmy SDP a LAO* a jejich aplikace

Oprimální postup prostorem stavů může být nalezen za použití algoritmu dynamického programování jako je strategie iterace (policy iteration) nebo iterace hodnoty (value iteration). Nevýhoda těchto metod dynamiického programování je, že prohledávají celý prostor stavů. Touto metodou nalezne algoritmus postup pro každý možný startovní stav. V kontrastu s tímto heuristický prohledávací algoritmus jako je např. A* nebo AO* řeší problém pro specifický startovní stav a použije vhodnou heuristiku ke zmenšení výseče (prostoru) pro vyhledávání a vynechá z uvažování regiony prostoru stavů, které nemohou být dosaženy ze startovního stavu optimálně. Toto je výhoda pro velké prostory stavů. Pro svou práci jsem se rozhodl zpracovat dva z algorutmů využívající dynamické programování k optimalizaci hodnoty a zároveň užívající heuristiku.

3.2.1 Směrované dynamické programování

Směrované dynamické programování(*Focussed dynamic programming*) je dle [17] algoritmus založený na heuristice a je určen k řešení omezených Markovských rozhodovacích procesů. V tomto algoritmu je zkombinován princip starších algoritmů pro řešení MRP a princip deterministického plánování.

Směrované dynamické programování (dále jen *SDP*) vychází ze dvou přístupů: první – využitý např v algoritmech *RTDP*, *EP*, nebo *LAO** využívá omezení souboru uvažovaných stavů a použití pouze těch které jsou nezbytně nutné k nalezení optimálního řešení. Tento přístup velmi omezuje prostor uvažovaných stavů a snižuje tím výrazně celkový počet aktualizací hodnot stavů. Druhý přístup, který je využit v PS, využívá postupu, při kterém se klade pozornost na pořadí, v němž budou stavy aktualizovány a tím zajišťují optimální

aktualizaci u každého jednotlivého stavu. Algoritmus se koncentruje na oblasti, ve kterých dochází k největším aktualizacím hodnot.

SDP využívá prvního přístupu k limitování počtu zkoumaných stavů pomocí heuristiky a druhého přístupu k zajištění nejvyšší efektivity aktualizace hodnot stavů. Zpětný algoritmus postupuje z cílového stavu směrem k startovnímu. Tuto metodu využívá např. Algoritmus A*. Vybírá stavy k aktualizaci pomocí heuristického odhadu hodnoty a heuristické ceny z aktuálního do cílového stavu. Připojením heuristických odhadů hodnoty stavů pomáhá obsáhnout stavy, které mohou mít nízké optimální hodnoty, ale zároveň mají vysoké aktuální hodnoty, které aktualizuje. Připojení heuristické ceny do startovního stavu zvýhodňuje stavy, které jsou vhodné, tedy mají největší vliv na hodnotu startovního stavu. Všechny stavy jsou při inicializaci nastaveny na nekonečně velké hodnoty. Hodnota stavu je v každém okamžiku na horní hranici optimální hodnoty stavu. Algoritmus udržuje prioritní frontu stavů, které mají být aktualizovány a jsou řazené tak, že stav na nejvyšším vrcholu fronty má nejmenší klíčovou hodnotu. Klíčová hodnota se získává součtem heuristické ceny ze startu do aktualního stavu *s* a (kontinuálně aktualizované) heuristické ceny ze stavu *s* k cíli. Fronta je inicializovaná (tedy je do ní vložen) cílovým stavem, jehož klíčová hodnota je heuristická cena ze startovního stavu.

Po vyjmutí fronty je hodnota stavu *s* a jeho sousedů aktualizována. Každý z těchto aktualizovaných stavů spočítá svou vlastní hodnotu δ, kterážto je rozdílem hodnoty stavu před a po aktualizaci. Jestliže δ pro každý jednotlivý stav je větší než námi zvolená prahová mez ϵ , je tento předán zpět do fronty s nově vypočtenou klíčovou hodnotou. Pokud již takový stav ve frontě existuje, porovnává se hodnota jejich klíčových stavů. Pokud klíčová hodnota nového stavu je menší než hodnota stavu ve frontě, je starý stav ve frontě vyjmut a do fronty přidán stav s menší klíčovou hodnotou. Hodnoty δ jsou určeny pouze k rozhodnutí zda bude stav vložen do fronty.

Algoritmus SDP

1. Vyjmutí stavu z frony. Stav označit x .
2. Pro stav $s \in (x \, U \, předchůdce(x))$
   - 2.1 $V'(s) := \min_{a \in A \, (sj \in s)} \Sigma \; P(s, a, s_j) * (C(s, s_j) + V(s_j))$
   - 2.2 $\Delta := |V(s) - V'(s)|$
   - 2.3 $V(s) := V'(s)$
   - 2.4 Jestliže $\Delta > \epsilon$
     - 2.5.1 $H(r, s) :=$ *Heuristická cena ze startovního stavu do stavu s*
     - 2.5.2 $G(s, g) :=$ *Heuristická cena ze stavu s k cíli*
     - 2.5.3 $K := H(r, s) + G(s, g)$
     - 2.5.4 Vložit stav s do fronty s klíčovou hodnotou K

3.2.2 Algoritmus LAO* a jeho modifikace

Algoritmus *LAO** popsaný v [18] vychází ze známého heuristického prohledávacího algoritmu pro prohledávání prostoru stavů AO*. AO* nachází řešení, které má podmíněnou strukturu a tvoří strom, nebo častěji acyklický graf. Rozdílem LAO* od AO* je skutečnost, že *LAO** využívá pro revizi hodnoty stavu metody dynamického programování strategie iterace nebo hodnotové iterace. Dalším rozdílem je, že LAO* využívá skoků, odkud vychází jeho název.

Algoritmus konstruuje explicitní graf G', který na počátku obsahuje pouze startovní

stav. Cípový nebo listový stav v tomto grafu je nazýván stavem terminálním pokud jde o cílový stav, jinak je tento stav neterminální. Cípový stav je takový stav, jež leží na hranici grafu nejlepšího řešení, tedy stímto grafem sousedí. Neterminální cípový stav může být expandován přidáním jeho následníků do explicitního grafu s výjimkou těch, které explicitní graf obsahuje.

Po expanzi se stav stává součástí grafu nejlepšího částečného řešení tak, že se připojí ke svým předchůdcům, kteří jsou součástí již existujícího nejlepšího řešení (tedy stavů, jejichž následováním lze dojít do cíle optimálně). Stav se pro expanzi vybírá tak, že to musí být cípový stav kolem grafu nejlepšího částečného řešení. Pokud existuje víc takových stavů, vybírá se podle nejmenší heuristické hodnoty. Dále, pokud je tento stav expandován, není již dále považován za stav cípový ale listový. Proto ho již nelze využít pro další expanzi.Tento stav je ale součástí souboru nejlepších částečných řešení, takže se může stát , že bude opět do grafu nejlešího řešení zařazen. Na expandovaný stav a na stavy, s nimiž tvoří nejlepší částečné řešení, je provedena hodnotová iterace nebo iterace strategie pro aktualizaci hodnoty stavů. Tento postup se opakuje do chvíle, dokud má graf nejlepšího řešení nějaké neexpandované cípové stavy. Pokud je splněna podmínka a pokud bylo k výpočtu využito stategie iterace, je výstupem algoritmu existující graf nejlepšího řešení. Pokud ale byla použita hodnotová iterace, musí se znovu provést na stavy grafu nejlepšího řešení. Iterace musí probíhat tak dlouho, dokud není splněna jedna ze dvou následujících podmínek:

1. Chybová mez musí být menší než prahová mez ϵ . Chybovou mezí se rozumí rozdíl mezi horní a dolní mezí optimální hodnoty startovního stavu. Aktuální cena stavu slouží jako dolní mez. Horní mezí je výsledek rovnice $f^{\pi}(s)+\phi^{\pi}(s)*r$, kde $f^{\pi}(s)$ je odhadovaná cena startovního stavu (t.j aktuální cena stavu), $\phi^{\pi}(s)$ znamená průměrný čas potřebný k dosažení cílového stavu ze startovního a $r=max_{i\in StartCíl}|f'(i)-f(i)|$ je Bellmanův zbytek z testu konvergence, kde $StartCíl\in S$ značí množinu stavů navštívenou grafem nejlepšího řešení.
2. Pokud se graf nejlepšího řešení změní během iterace tak, že má nějaké neexpandované cípové stavy, algoritmus skočí zpět na bod 2.

LAO algoritmus*

1. Graf G' na začátku obsahuje startovní stav s.
2. Pokud má graf s nejlepším řešením neterminální (cípové stavy) dělej:
   2.1 Expanze nejlepšího částečného řešení: Expanze nějakého neterminálního cípového stavu a přidání nových následníků do G'. Pro každý nový stav i přidaný expanzí stavu n do množiny G platí: Jestliže je to stav cílový pak $f(i):=0$ jinak $f(i):=h(i)$.
   2.2 Aktualizace ceny stavu a označení nejlepší akce:
      2.2.1 Vytvoření souboru Z obsahujícího expandovaný stav a všechny jeho předchůdce v explicitním grafu kolem označených buňek v prostoru.
      2.2.2 Provedení hodnotové iterace nebo iterace strategie na stavy v množině Z pro aktualizaci ceny stavu a označení nejlepší akce pro každý stav.
3. Test konvergence:
   Pokud byla použita iterace strategie poté skoč na krok 4., jinak proveď iterace hodnoty na stavy v grafu nejlepších řešení. Pokračovat dokud není splněno alespoň jedno z těchto kritérií: a) pokud chybová mez je

menší než ϵ jdi na krok 4, b) pokud se graf nejlepšího řešení změní tak, že obsahuje nějaké neexpandované cípové stavy pak jsi na krok 2

4.Extrakce grafu s nejlepším řešením.

*Modifikovaný LAO**

1. Graf G' na začátku obsahuje startovní stav s.
2. Pokud není nalezen cíl dělej:
   2.1 Expanze nejlepšího částečného řešení: Expanze nějakého neterminálního cílového stavu a přidání nových následníků do G'. Pro každý nový stav i přidaný expanzí stavu n do množiny G platí: Jestliže je to stav cílový pak $f(i) := 0$ jinak $f(i) := h(i)$
   2.2 Aktualizace ceny stavu a označení nejlepší akce:
   2.2.1 Vytvoření souboru Z obsahujícího expandovaný stav a všechny jeho předchůdce v explicitním grafu nejlepších řešení. Expandovanému stavu přiděléna váha.
   2.2.2 provedení hodnotové iterace na stavy v množině Z pro aktualizaci ceny stavu a označení nejlepší akce pro každý stav.
3. Extrakce grafu.

3.2.3 Hlavní části algoritmu SDP

- *Prioritní fronta* – pomocí prioritní fronty se vybírají stavy vhodné expanzi. Hodnoty vkládané do prioritní fronty se ukládají do objektu Tlist, který funguje jako alternativa dynamického pole. Hodnoty se do prioritní fronty vkládají za určitých podmínek pomocí jednoduchých cyklů if-else. Tento cyklus při vkládání kontroluje, zda stav o stejných souřadnicích již ve frontě neexistuje; pokud ano, je nahrazen novým jestliže tento nový stav má menší klíčovou hodnotu K, tedy vyšší prioritu. Pokud stav má nejnižší prioritu, je umístěn na vrchol fronty, naopak pokud má stav nejnižší prioritu je stav umístěn na konec této fronty, nebo alternativně to takového místa ve frontě, jemuž odpovídá jeho prioritní hodnota K.

- *Hodnota K* – tato hodnota je vedle prioritní fronty další důkežitou součástí tohoto plánovacího algoritmu. Je složena ze dvou heuristik heuristiky start - aktuální stav H a heuristiky aktuální stav - cíl G. Tato hodnota určuje polohu stavu v prioritní frontě, neurčuje však jestli tento stav bude vůbec do fronty vložen.

- *Heuristika startovní stav-aktuální stav H* – tato heuristika je využita jako první složka výpočtu prioritní hodnoty K. Její implementace je jednoduchá: skládá se z polohy startovního stavu a aktuálního stavu, pomocí kterých je vypočtena vzdálenost mezi těmito dvěma body. Prototože je vzhledem k velikosti pole tato hodnota v porovnání s druhou v rovnici výpočtu klíčové hodnoty nevyvážená, je tato hodnota vynásobena 10^4 . Pro výpočet využívá jednoduchého vzorce pro Euklidovskou vzdálenost:

$$H(s,a) = \sqrt{(s_x - a_x)^2 + (s_y - a_y)^2}$$

- *Heuristika aktuální stav-cíl* - tato heuristika je druhou částí rovnice prioritní hodnoty K. Je určena k odhadu budoucího stavu. Princip zpočívá ve výpočtu heuristické hodnoty ze všech možných akcí z tohoto stavu. K výpočtu je využita využita stejná rovnice, pomocí které vypočítáváme hodnotu stavu v hlavním algoritmu. Rovnice k výpočtu heuristiky je následující:

$$G_a(a,c) = \sum_{s_j \in S} P(s_i, a, s_j) * (C(s_i, s_j) + V_{(a,s)}(s_j))$$

Výsledná hodnota heuristiky je stanovena minimální hodnotou ze všech možných akcí:

$$G(s,c) = \min_{a \in A} G_a(s,c)$$

- *Překážky a jejich obcházení* - překážky jsou v poli generovány generátorem překážek. Algoritmus je přizpůsoben tak, aby při navštívení stavu obsahujícího překážku byla ceně přidělena velmi vysoká hodnota. To ve výsledku způsobí, že tento stav je umístěn hluboko do prioritní fronty a přednost mají ty stavy které překážku neobsahují.

- *Pravděpodobnost přechodu* – pravděpodobnosti přechodu mohou být zadávané uživatelem.

- *Cena* – cenová funkce se vypočítává pomocí výpočtu Euklidovské vzdálenosti aktuální_stav-start jako v algoritmu MLAO*. Autoři práce[17] sice uvažují prostor stavů s náhodnými cenami, já jsem zvolil postup popsaný v metodě LAO*. Tím jsem se snažtl dosáhnout vyšší přímočarosti algoritmu na cestě do cíle.

3.2.4 Hlavní části modifikovaného algoritmu LAO*

Fronta G – tato fronta obsahuje všechny prozkoumané cípové stavy, které vznikly zvažováním částečných řešení. Do této fronty se po každé expanzi cípového stavu ukládají stavy nové.

Fronta CRG – tato fronta obsahuje všechny stavy které jsou částečných řešení grafu. Při postupu algoritmu prostorem jsou do této fronty zahrnovány nově expandované stavy naazujicí na najlepší částečné řešení.

Cena - cena je spočtena výpočtem Euklidovské vzdálenosti od aktuálního stavu do cíle.

Pravděpodobnost přechodu – pravděpodobnosti přechodu jsou zadávané uživatelem. Jejich hodnota je za účelem srovnání s algoritmem SDP totožná s tímto algoritmem. Je užitá při výpočtu hodnoty stavu stejně jako cena.

Expanze – Při expanzi se vybere stav z množiny G a je expandován: tj. je zkoumáno jeho okolí a do množiny G jsou přidány jen ty stavy které nejsou překážkami, nejsou mimo pole a nejsou to již stavy expandované, respektive nejsou to stavy, které jsou součástí nejlepšího částečného řešení

Výpočet stavu k expanzi – nový stav se jednoduše vybere z množiny stavů G, nově

expandovaný stav je stav cípový: tj. není v množině částečného řešení ani nebyl dosud expandován.Jjeho další vlastností je, že se musí jednat o stav který je nejvýhodnější – tj. jeho heuristická vzdálenost k cíli je co nejkratší. Tímto způsobem se zajistí vybíraní pouze cípových stavů umžňujících nejrychlejší prostup prostorem. Jako jeho hodnota je zvolena heuristická vzdálenost od startu do cíle.

Vytvoření grafu K obsahujícího nově expandovaný stav a stavy mířící ke startovnímu stavu – expandovaný stav se pomocí algoritmu připojí ke grafu optimálního řešení. To je provedeno tím způsobem, že nově expandovaný stav zkoumá své okolní stavy. Pokud existují stavy, které jsou součástí fronty CRG, potom vybírá z těchto stavů ten s nejnižší hodnotou. Tím zajistí, že se jedná o stav nejbližší cíli.

Provedení hodnotové iterace na stavy v množině K - na stavy nejlepšího částečného řešení je provedena hodnotová iterace. Za ϵ je nastavena stejně u algoritmu SDP na hodnotu 2000.

4 IMPLEMENTACE NAVRŽENÉHO ŘEŠENÍ

Algoritmy jsem zpracoval v prostředí Delphi 7, což je nadstavba jazyka Object Pascal. K tomuto rozhodnutí mě vedla zkušenost tímto prostředím.

4.1 Zpracovaný program

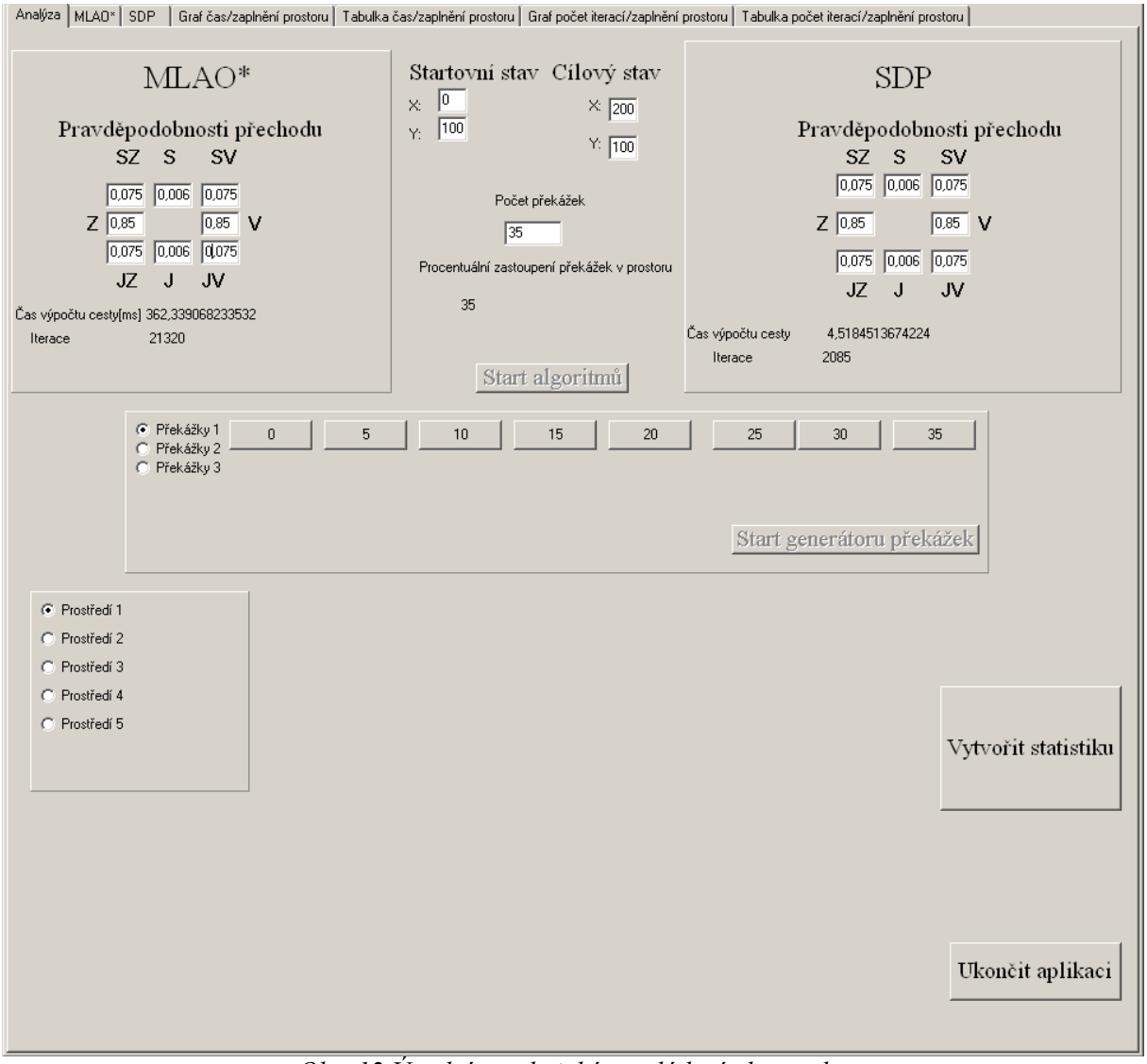
Program umožňuje snadnou interpretaci výstupů mapou cesty robota, a porovnání rychlosti obou algoritmů, kterou umožnuje přehledná tabulka a graf. Součástí porovnávací funkce jsou také soubory, které se vytvářejí na základě dat z programu. Jako prostředí jsem zvolil mřížku vzhledem k tomu, že oba algoritmy vychází z heuristických prohledávacích grafových metod, jež jsou často uplatňovány na mřížce. Vzhledem k tomu, že jsem se primárně zabýval algoritmem SDP, který je určen pro venkovní prostředí, mým cílem bylo generovat překážky simulující takové prostředí. U algoritmů se poté počítá s tím že budou procházet napříč mapou, z jednoho okraje mapy do druhého.

4.2 Užité algoritmy

V programu je implementován algoritmus SDP. Pro porovnání tohoto algoritmu byl zaimplementován MLAO* což je jednoduchá modifikace algoritmu LAO*. Tento algoritmus jsem zvolil proto, že také využívá dynamické programování. Tento algoritmus jsem také modifikoval (více v předcházejcím popisu jednotlivých metod). Oba algoritmy v programu využívají metody Markova rozhodovacího procesu, konkrétně se jedná o metodu hodnotové iterace. Použité objektově orientované prostředí Delphi umožňuje využít jeho výhod tak, že některé procesy, jako například práce s hodnotami v poli umožňuje sdílení pro oba algoritmy. Program je formálně rozdělen do tří částí: *Jádro*, ve které jsou obsaženy funkce a procedury pro obcházení překážek, výpočtem klíčových hodnot atd. Do druhé části nazvané nazvané *Fronta* jsem umístil procedury pro práci s prioritní frontou pro SDP a frontou pro LAO*. Poslední část *Typy*, obsahuje zbylé komponenty pro funci programu.

4.3 Program a jeho ovládání

Ovládání programu je velmi jednoduché. Uživatel zadá polohu startovního a cílového bodu (zpravidla mění jen hodnoty y), vybere prostředí z menu, v případě prostředí č.1 a č.2 může zvolit odlišnou velikost překážek. Volbou přibližné hodnoty zaplněnosti tlačítky z menu se mu objeví možnost generovat mapu. Jinou možností je, pokud chceme vyšší zaplněnost, zvolit jakoukoli pomocí tlačítka a pak ručne vložit požadovanou hodnotu do editačního okénka. Potom zbývá jen stisknout tlačítko start algoritmů. V případě potřeby může pro prostředí č.1 nebo č.2 pořídit statistické údaje jednoduchou volbou velikosti překážek, typu prostředí a stiskem tlačítka *Vytvořit statistiku*. Volba vytvořit statistiku je možná pouze v případě generování prostředí č.1 a č.2 a velikosti překážek malé a střední. Uživatel má také možnost zadávat předdefinované množtví překážek ve třech různých velikostech. Po zvolení počtu překážek buď tlačítkem nebo přímo v editačním okénku spustí uživatel program. Algoritmus nageneruje přkážky, případně je vykreslí dle zvoleného vzoru. Uživatel má možnost získat informace o rychlosti algoritmu, a počtu opakování každého algoritmu.



Obr. 12 Úvodní, analytická a ovládací obrazovka

Program dále umožňuje vytvořit statistiku, při které se náhodně generuje prostor uživatelem zvoleným velikostí překážek. Algoritmus postupně provádí generování a výpočet cesty prostorem, vždy desekrát u každého procentního množství překážek. Na základě těchto údajů se vypočte aritmetický průměr těchto jednotlivých opakování a zapisuje se do tabulek a grafu. Výstupem statistiky jsou textové soubory, které lze dále zpracovávat.

Na základě těchto údajů se vypočte aritmetický průměr těchto jednotlivých průchodů prostorem a zapisuje se do grafu.

4.3.1 Hlavní části aplikace

Aplikace je rozdělena do panelu. V hlavním panelu s názvem *Analýza* má možnost uživatel přehledně ovládat program. Je rozdělen do několika částí.

VRCHNÍ MENU



Obr. 13 Horní část menu s lištami

V horní části menu má uživatel možnost přejít na další panely.

- Panel MLAO* - tento panel slouží k zobrazení scény algoritmu MLAO*
- Panel SDP – stejně jako předchozí s tím rozdílem, že zobrazuje SDP
- Graf čas/zaplnění prostoru – zobrazuje statistiku časové náročnosti jednotlivých algoritmů v závislosti na zaplnění prostoru překážkami
- Graf počet iterací/zaplnění prostoru – zobrazuje závislost počtu iterací na postupně se zvyšujícím zaplnění prostoru.
- Tabulka čas/zaplnění prostoru – zobrazuje výpis 10 průchodů prostorem při zvyšujícím se počtu překážek.
- Tabulka počet iterací/zaplnění prostoru – stejně jako předchozí tato tabulka zobrazuje počet opakování v závislosti na zaplnění prostoru překážkami. Data z obou tabulek jsou pak užita jako podklad do výstupních souborů.

MLAO*

	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
0procent	20099	20099	20099	20099	20099	20099	20099	20099	20099	20099
5procent	20705	20099	20502	20705	20705	20099	20502	20300	20099	20300
10procent	20099	21320	20502	20909	20099	20099	21735	20300	20909	20705
15procent	20705	20909	22365	22577	21114	20909	22365	21114	20502	20909
20procent	21735	21735	22790	21944	21320	22365	21114	21735	21944	21320
25procent	23219	22577	20705	21320	22790	23004	21735	20705	21320	21735
30procent	21114	21944	22154	23219	21114	21735	20300	21944	26795	22365
35procent	23219	23652	21320	21735	22365	22365	21735	23004	21735	21320

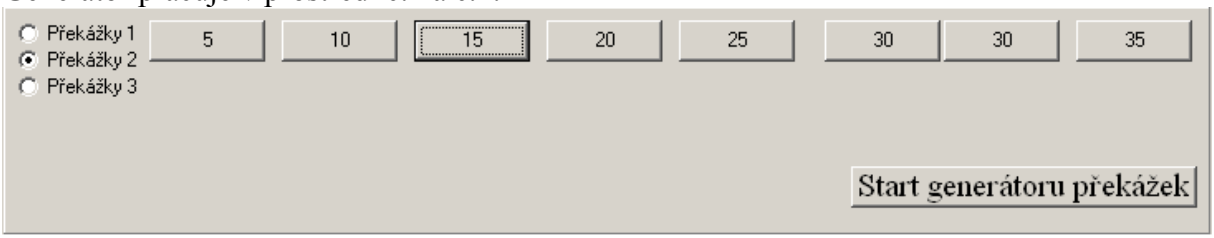
SDP

	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
0procent	2004	2004	2004	2004	2004	2004	2004	2004	2004	2004
5procent	2023	2004	2052	2022	2022	1990	2013	1996	1986	2031
10procent	2004	2071	2117	2013	1999	2004	2022	2062	2022	2026
15procent	2101	2077	2080	2018	2039	2054	2063	2050	2069	2067
20procent	2100	2031	2035	2063	2087	2023	2034	2088	2064	2021
25procent	2095	2089	2040	2115	2034	2105	2256	2009	2090	2075
30procent	2012	2004	2046	2057	2089	2144	2035	2025	2009	2046
35procent	2063	2089	2092	2072	2118	2043	2136	2084	2142	2085

Obr. 14 Ukázka vygenerované tabulky v programu

GENERÁTOR PŘEKÁŽEK

V této části uživatel volí přibližnou procentuální zaplněnost prostoru překážkami. Generátor pracuje v prostředí č.1 a č.2.



Obr. 15 Panel generátoru překážek

Uživatel může volit ze tří typů překážek Může volit ze tří typů velikostí překážek: malých, středních a velkých.Malé překážky mají velikost 2*2 políček pro čtvereček a 4*2 políček pro obdélník, střední překážky jsou o velikosti 8*8, a 16*8, velké překážky o velikosti 20*20 a 40*20. Uvedený algoritmus umožňuje zaplnit prostor o velikosti 200*200 políček objekty v závislosti na zvoleném prostředí.

SPODNÍ ČÁST OBRAZOVKY

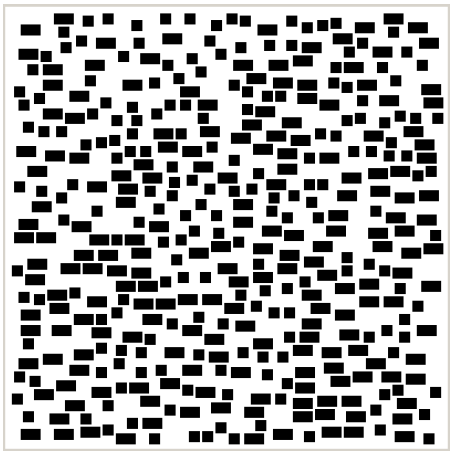


Obr. 16 Spodní část panelu Analýza. Zde má uživatel na výběr z prostředí, vytvořit statistiku a ukončit program

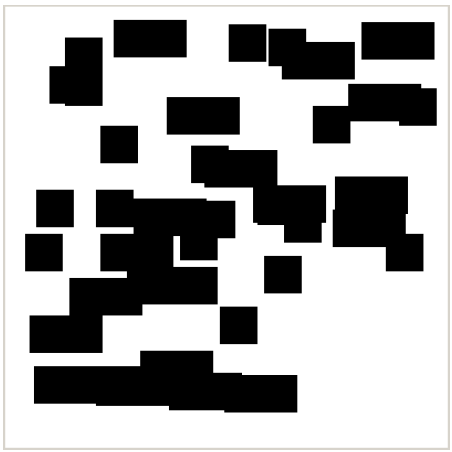
V této části má uživatel možnost tří voleb: zvolit si typ prostředí, anayzovat nebo ukončit aplikaci.

- *Typ prostředí*: jako prostředí ve kterém se robot pohybuje lze vybrat z několika možností:
-

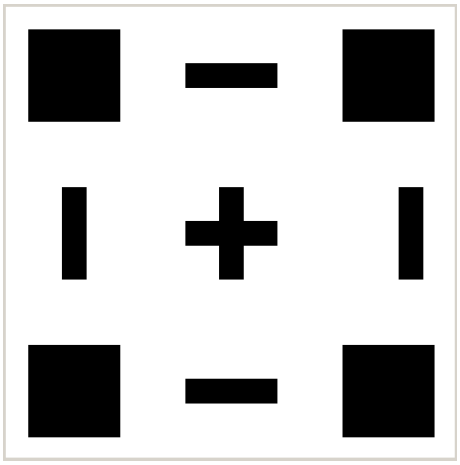
1. Jedná se o prostředí, ve kterém se generují překážky tak, aby se nedotýkaly. Algoritmus dodržuje odstup při vkládání obektů i od startovního a cílového stavu. Vytváří v daném diskrétním prostředí vkládá překážky v podobě čtverců a obdélníků. Je navržen tak, že při vkládání překážek ignoruje blízké okolí startovního a cílového stavu a počíná si tak, aby se každý následující objekt nedotýkal objektu vloženého před ním. Každý objekt má kolem sebe jakýsi „ochranný prostor“ o velikosti jednotky stavového prostoru. Maximální procentuální zaplnění prostoru tímto nastavením je 40 procent.
2. V tomto prostředí stejně jako algoritmus v prostředí prvním generuje náhodně překážky. Rozdílem je omezení jejich dotyku, v tomto prostředí se mohou překážky volně překrývat. S touto vlastností má algoritmus možnost vykreslovat velmi zajímavá nepravidelná prostředí, tvořící např. shluk překážek apod.
3.,4.,5. Tato prostředí již nejsou generována ale jsou pevně stanovena. Umožnují otestovat schopnost algoritmu vyrovnat se s překážkou nebo překážkami v různých modelových prostředích.



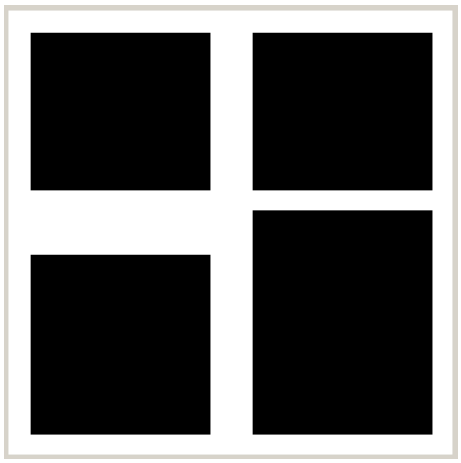
Obr. 17 Typ prostředí 1



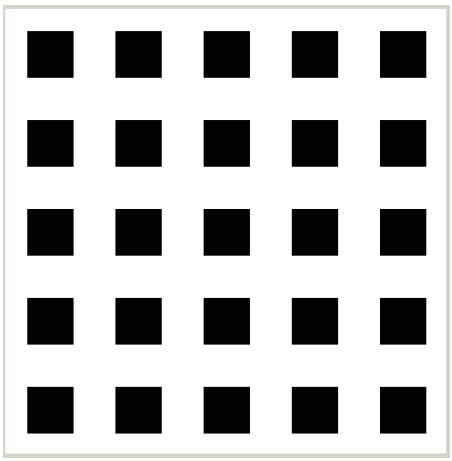
Obr. 18 Typ prostředí 2



Obr. 19 Typ prostředí 3



Obr. 20 Typ prostředí 4

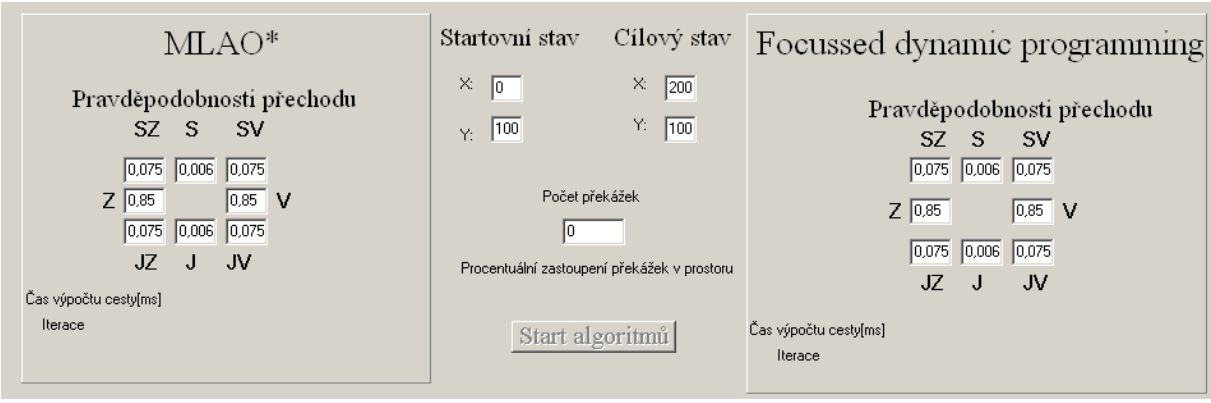


Obr. 21 Typ prostředí 5

- *Tlačítko vytvořit statistiku:* tato volba umožňuje uživateli možnost testovat algoritmy v daném prostředí. Touto volbou uživatel získá statistiku o jednotlivých opakováních. Podle počtu překážek algoritmus opakuje deset kroků (u velikosti překážek malé a střední) a 4 kroky u překážek velkých. Algoritmus generuje náhodné pole a vypočítává optimální plán cesty. Každý krok znamená obecný počet překážek oscilujíccí kolem zadaného procentuálního podílu překážek v prostoru. Užívá se na prostředí č.1 a prostředí č.2. Umožňuje uživateli porovnat oba algoritmy a to z hlediska počtu opakování a času výpočtu v milisekundách v náhodném prostředí. Výstupem jsou grafy a tabulky přímo v aplikaci a také textové výstupy které jsou dále použitelné
- *Tlačítko ukončit aplikaci* -ukončení aplikace

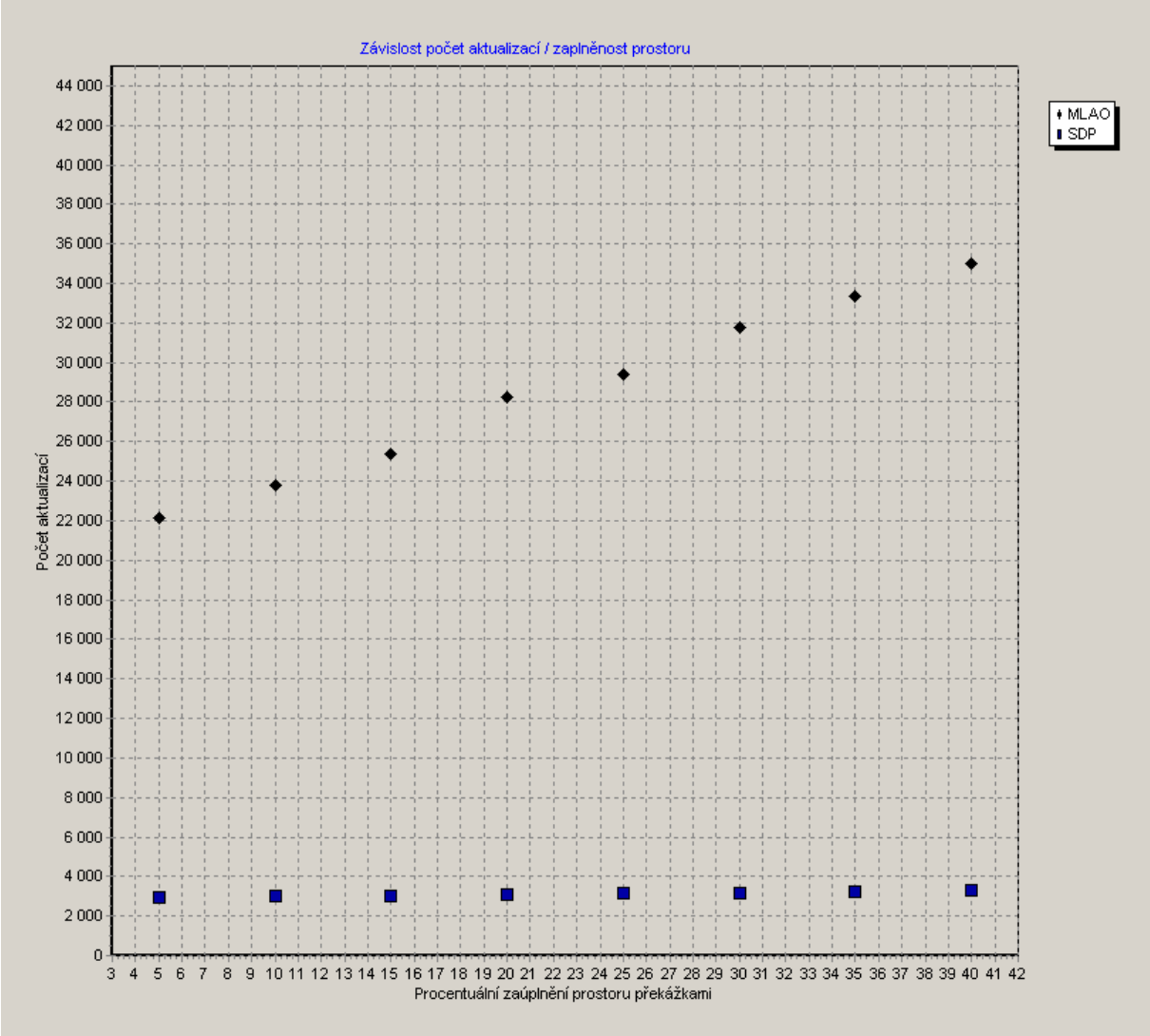
HORNÍ ČÁST PANELU ANALÝZA

Tato část slouží k zadávání údajů jako jsou poloha startu a cíle, nebo pravděpodobnosti přechodu obou algoritmů. Uživatel zde zadává pravděpodobnosti přechodu v prostoru, čímž ovlivňuje rychlost prostupu prostorem. Pro srovnání algoritmů jsou tyto hodnoty nastaveny na stejnou úroveň. Jelikož byly algoritmy postaveny na myšlence, že bude srovnáván jejich prostup prostorem pouze za situace startovní stav x:0,y:100, cílový stav x:200,y:100, jsou tomu také přizpůsobeny předem nastavené pravděpodobnosti. Do okénka počet překážek zadáváme počet, který bychom chtěli generováním dosáhnout. Pod tímto editačním okénkem se po generování zobrazí výsledná procentuální zaplněnost prostoru.



Obr. 22 Horní část panelu s možností volby startovního a cílového stavu a pravděpodobností přechodu.

Tento procentuální podíl má vzhledem k povaze prostředí své limity. Limitní počet pro malé překážky a za užití prostředí č. 1 je kolem 40 procent zaplněnosti, u dalších dvou velikostí překážek dostáváme podobné hodnoty. Jiná situace nastává při užití prostředí č. 2. Zaplněnost prostoru překážkami může být až 80 procent (mapa je v tomto případě zcela zaplněna překážkami, zbývají jen uzké pruhy u okrajů).



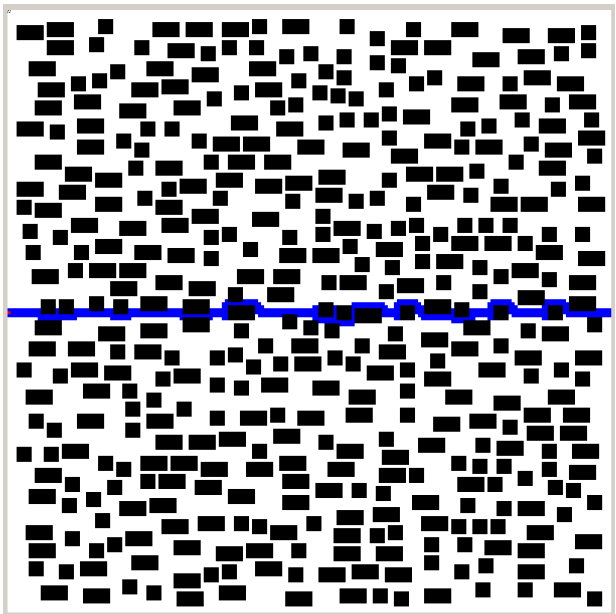
Obr. 23 Obrázek vygenerovaného grafu počet aktualizací v závislosti na zaplněnosti prostoru. MLAO diamanty, SDP čtverečky*

5 TESTY A SROVNÁNÍ ALGORITMŮ

Pro srovnání obou algoritmů byla užit prostor 200*200 stavů, ve kterém jsou náhodně generovány překážky ve tvaru čtverců a obdelníků. Pravděpodobnosti přechodu zadává uživatel. Testy a podmínky testů jsou uvedeny níže v tabulce. Testován byl čas výpočtu algoritmu a počet aktualizací hodnot každého algoritmu a to v závislosti na procentuálním obsazení překážek v prostoru. K dispozici je výše zmíněné prostředí, ve kterém je při experimetech prohazována poloha startovního a cílového stavu ve vodorovném a diagonálním směru tak, aby byla zhodnocena výpočetní náročnost a rychlost každého z algoritmů v daném prostředí. Při každém testu byly na daném prostředí vypočteny cesty pomocí obou algoritmů, desetkrát pro každý procentuální podíl překážek v prostoru. Jak již bylo řečeno v předcházejcím textu, prahová hodnota ϵ je nastavena na hodnotu 2000. Pravděpodobnosti přechodu jsou nastaveny pro oba algoritmy stejně.

- Test 1: poloha startu a cíle jsou naproti sobě vodorovně, testuje se počet iterací a čas výpočtu při zvyšujícím se počtu překážek v prostředí č. 1. za použití překážek č.1. Poloha startu: x:0,y:100, cíle x:200,y:200.
- Test 2: poloha startu a cíle je ale v mapě umístěna diagonálně: testuje se postupně iterace a čas výpočtu za zvyšujícího se počtu překážek prostředí č. 1. za použití překážek č.1.
- Test 3: poloha startu a cíle jsou naproti sobě vodorovně , testuje se počet iterací a čas výpočtu při zvyšujícím se počtu překážek v prostoru prostředí č. 2., za použití překážek č. 1. Poloha startu: x:0,y:100, cíle x:200,y:100.
- Test 4: poloha startu a cíle je diagonální , testuje se počet iterací a čas výpočtu při zvyšujícím se počtu překážek v prostředí č. 2, za použití překážek č. 2. Poloha startu: x:0,y:0, cíle x:200,y:200.

Test 1.



Obr. 24 Modelové prostředí a příklad vytvořené cesty v testu č.1

Iterace SDP										
	1	2	3	4	5	6	7	8	9	10
0.procent	2004	2004	2004	2004	2004	2004	2004	2004	2004	2004
5.procent	2013	2040	2004	2009	2024	2009	2033	2045	2023	2013
10.procent	2066	2000	2030	2014	2064	2013	1995	2023	2051	2076
15.procent	2051	2073	2052	2032	2028	2126	2081	2109	2043	2022
20.procent	2106	2030	2089	2019	2028	2044	2035	2125	2002	2079
25.procent	2025	2065	2086	2057	2090	2033	2058	2024	2100	2080
30.procent	2068	2162	2062	2167	2098	2032	2118	2175	2114	2044
35.procent	2224	2131	2120	2043	2192	2037	2155	2258	2032	2033

Iterace MLAO*										
	1	2	3	4	5	6	7	8	9	10
0.procent	20099	20099	20099	20099	20099	20099	20099	20099	20099	20099
5.procent	20502	20502	20099	20502	20502	20099	21527	21114	20705	21114
10.procent	20099	20300	20705	20705	20300	20909	21320	21114	21114	22154
15.procent	21944	22154	21735	20909	21114	20705	20909	21320	21527	21320
20.procent	21735	23435	21114	23435	22577	21944	22365	21735	21944	23004
25.procent	22790	20705	23435	21944	21320	21527	22154	21944	22154	22577
30.procent	22365	24089	24089	23435	22154	22577	21527	23435	21944	20909
35.procent	22577	22577	23004	21527	22365	21944	22790	21944	21944	22365

Čas SDP[ms]										
	1	2	3	4	5	6	7	8	9	10
0.procent	4,2	4,0	3,9	4,2	3,9	3,9	4,2	4,0	3,9	4,6
5.procent	7,7	4,3	4,2	4,3	4,2	3,9	4,4	5,0	4,1	4,3
10.procent	4,1	5,3	4,4	4,3	4,0	4,2	4,1	4,1	4,2	4,5
15.procent	4,8	4,7	4,7	4,5	4,6	4,6	4,9	4,8	5,0	4,6
20.procent	4,8	4,6	4,8	4,6	4,7	4,5	4,5	4,8	4,6	4,7
25.procent	4,3	4,6	4,9	4,7	5,0	4,2	4,3	8,4	4,5	4,9
30.procent	4,5	4,9	4,7	4,8	4,7	4,5	4,6	4,7	10,6	4,7
35.procent	5,0	4,9	4,5	4,4	4,9	4,2	4,6	5,0	4,3	6,0

Čas MLAO*[ms]										
	1	2	3	4	5	6	7	8	9	10
0.procent	337,1	357,7	363,5	362,6	429,5	371,0	346,5	345,4	343,4	356,0
5.procent	735,2	923,9	332,3	334,0	335,1	335,7	397,5	379,2	382,4	383,5
10.procent	361,1	371,7	385,4	377,4	349,2	589,8	366,0	347,7	349,9	380,8
15.procent	384,4	382,3	394,7	378,7	402,8	351,2	355,7	370,6	388,7	429,2
20.procent	388,3	417,0	346,0	424,3	414,7	410,6	414,2	1181,0	376,1	388,8
25.procent	396,0	333,9	403,7	365,3	425,1	360,7	396,3	403,7	410,4	430,7
30.procent	387,2	437,6	416,7	400,0	381,0	399,4	457,1	449,1	393,3	371,4
35.procent	372,7	372,1	407,8	354,9	420,4	389,0	383,6	363,7	676,5	661,1

Iterace SDP		
	Průměr	Nejlepší
0.procent	2004	2004
5.procent	2021,3	2004
10.procent	2033,2	1995
15.procent	2061,7	2022
20.procent	2055,7	2002
25.procent	2061,8	2024
30.procent	2104	2032
35.procent	2122,5	2032

Iterace MLAO*		
	Průměr	Nejlepší
0.procent	20099	20099
5.procent	20666,6	20099
10.procent	20872	20099
15.procent	21363,7	20705
20.procent	22328,8	21114
25.procent	22055	20705
30.procent	22652,4	20909
35.procent	22303,7	21527

Čas SDP[ms]		
	Průměr	Nejlepší
0.procent	4,069	3,876
5.procent	4,630	3,908
10.procent	4,331	4,032
15.procent	4,713	4,481
20.procent	4,666	4,473
25.procent	4,977	4,194
30.procent	5,270	4,511
35.procent	4,769	4,248

Čas MLAO*[ms]		
	Průměr	Nejlepší
0.procent	361,281	337,146
5.procent	453,883	332,299
10.procent	387,886	347,650
15.procent	383,831	351,232
20.procent	476,099	345,985
25.procent	392,583	333,883
30.procent	409,280	371,393
35.procent	440,167	354,922

Test 2.

Iterace SDP										
	1	2	3	4	5	6	7	8	9	10
0.procent	2680	2680	2680	2680	2680	2680	2680	2680	2680	2680
5.procent	2735	2685	2696	2710	2675	2729	2690	2690	2676	2775
10.procent	2700	2764	2706	2715	2744	2741	2750	2801	2750	2725
15.procent	2761	2789	2756	2842	2762	2852	2820	2894	2789	2745
20.procent	2851	2950	2909	2804	2893	2868	2936	2809	2850	2863
25.procent	2825	2950	2897	2910	2851	2944	2773	2902	2827	2932
30.procent	2836	2963	2882	3094	2937	2940	2866	2893	3001	2898
35.procent	2933	3000	3003	2900	3076	2957	2985	2989	3022	2963

Iterace MLAO*										
	1	2	3	4	5	6	7	8	9	10
0.procent	20099	20099	20099	20099	20099	20099	20099	20099	20099	20099
5.procent	21320	21320	22154	21944	21114	22154	20909	20502	21735	22577
10.procent	23219	22577	26334	22365	22365	22154	23870	22790	24309	24530
15.procent	25424	24309	24309	23435	25877	24309	25877	25877	24752	24975
20.procent	27965	23435	27965	26105	29402	26564	27965	29160	26564	26334
25.procent	29402	26105	28919	29402	27729	27729	28919	27965	30134	28679
30.procent	27494	30134	34452	29645	31374	26564	29402	34979	29889	29645
35.procent	30627	27494	34715	34979	34452	33410	30627	31124	30380	30134

Čas SDP[ms]										
	1	2	3	4	5	6	7	8	9	10
0.procent	9,4	7,8	8,1	8,6	12,8	8,7	8,7	8,4	8,6	8,5
5.procent	9,2	8,4	8,4	8,9	9,3	8,7	9,9	9,1	9,9	9,0
10.procent	8,9	9,6	9,4	8,7	8,8	8,5	8,4	9,5	9,3	8,3
15.procent	9,0	9,8	10,1	9,5	10,8	11,1	9,8	102,7	9,1	9,2
20.procent	10,0	9,9	9,0	11,6	8,0	15,8	9,4	25,8	8,8	8,8
25.procent	7,9	8,0	8,5	16,6	7,9	8,1	7,5	11,4	9,6	7,9
30.procent	16,4	9,2	12,2	8,4	8,1	8,2	7,7	8,1	8,4	7,8
35.procent	7,8	8,1	7,9	7,9	8,1	8,1	7,9	8,0	7,8	12,4

Čas MLAO*[ms]										
	1	2	3	4	5	6	7	8	9	10
0.procent	502,3	375,0	382,5	391,3	387,9	462,6	387,0	389,4	381,2	372,1
5.procent	387,9	388,1	434,3	414,8	530,4	565,2	574,4	513,9	418,6	482,5
10.procent	513,0	536,6	593,9	478,5	468,4	463,2	462,6	844,1	481,2	476,4
15.procent	518,4	509,2	485,6	491,0	609,0	531,0	600,3	1231,7	626,5	555,6
20.procent	631,7	475,0	637,7	599,9	682,1	1094,9	918,0	812,5	618,2	659,3
25.procent	778,6	660,1	1336,2	859,6	852,9	801,1	740,9	651,9	776,4	854,3
30.procent	1024,5	688,2	798,8	679,1	701,8	517,8	613,9	844,9	684,1	621,2
35.procent	624,2	546,6	896,7	1183,2	817,7	802,7	650,9	678,4	1448,3	655,6

Iterace SDP		
	Průměr	Nejlepší
0.procent	2680	2680
5.procent	2706,1	2675
10.procent	2739,6	2700
15.procent	2801	2745
20.procent	2873,3	2804
25.procent	2881,1	2773
30.procent	2931	2836
35.procent	2982,8	2900

Iterace MLAO*		
	Průměr	Nejlepší
0.procent	20099	20099
5.procent	21572,9	20502
10.procent	23451,3	22154
15.procent	24914,4	23435
20.procent	27145,9	23435
25.procent	28498,3	26105
30.procent	30357,8	26564
35.procent	31794,2	27494

Čas SDP[ms]		
	Průměr	Nejlepší
0.procent	8,956	7,754
5.procent	9,062	8,356
10.procent	8,937	8,338
15.procent	19,103	8,981
20.procent	11,702	7,985
25.procent	9,338	7,532
30.procent	9,450	7,710
35.procent	8,387	7,754

Čas MLAO*[ms]		
	Průměr	Nejlepší
0.procent	403,125	372,061
5.procent	471,017	387,916
10.procent	531,789	462,615
15.procent	615,835	485,572
20.procent	712,929	474,958
25.procent	831,209	651,941
30.procent	717,432	517,843
35.procent	830,443	546,566

TEST 3.

Tento test v prostředí č. 2 již ukazuje především vyšší časovou náročnost MLAO* v porovnání s testem č.1.

Iterace SDP										
	1	2	3	4	5	6	7	8	9	10
0.procent	2004	2004	2004	2004	2004	2004	2004	2004	2004	2004
5.procent	2004	2004	2017	2017	2004	2017	1999	2031	2022	2022
10.procent	2035	2033	2022	2036	2036	2049	2109	2026	2231	2042
15.procent	2031	2062	2000	2074	2109	2194	2051	1990	1971	2121
20.procent	2816	2293	2054	2909	2006	2805	2695	2128	2506	2092
25.procent	2070	2419	2105	2167	2123	1979	2147	2046	2109	2946
30.procent	2081	2292	2090	2161	2536	2701	2546	2850	3820	3630
35.procent	4209	2216	4161	2100	2560	2861	2639	3005	8896	2632

Iterace MLAO*										
	1	2	3	4	5	6	7	8	9	10
0.procent	20099	20099	20099	20099	20099	20099	20099	20099	20099	20099
5.procent	20502	20099	20099	20502	20099	20705	20099	20300	20909	20705
10.procent	20705	20502	20099	21114	21735	20300	20705	20502	21527	22577
15.procent	20705	20909	20099	25199	23870	20300	20705	20099	20099	24309
20.procent	24309	23652	20909	25199	21320	21944	26564	22790	35244	21320
25.procent	21114	24309	21944	21114	21944	20099	24309	21735	24089	24975
30.procent	25650	24089	22154	29889	23435	24975	36314	27729	43659	25199
35.procent	30875	26334	33152	23004	42485	66065	23004	57969	63902	31124

Čas SDP[ms]										
	1	2	3	4	5	6	7	8	9	10
0.procent	4,3	3,9	4,1	4,3	4,0	3,9	4,1	4,3	4,0	4,0
5.procent	4,0	3,5	4,2	4,2	4,0	3,8	3,9	5,4	4,9	3,9
10.procent	4,3	7,2	3,8	3,9	4,2	4,0	4,5	51,1	4,5	4,2
15.procent	4,7	4,6	4,4	4,5	4,8	5,0	4,2	7,9	4,4	4,8
20.procent	11,4	4,9	4,6	6,0	4,5	6,8	5,8	4,9	5,6	8,9
25.procent	8,9	9,8	4,8	9,1	4,8	4,9	4,5	4,8	4,4	5,7
30.procent	4,2	5,9	4,3	4,7	5,6	5,7	5,2	5,7	7,7	7,4
35.procent	8,7	5,1	8,2	35,8	5,2	6,0	5,8	7,2	13,5	5,6

Čas MLAO*[ms]										
	1	2	3	4	5	6	7	8	9	10
0.procent	323,2	453,8	464,4	475,7	430,2	673,8	357,9	777,7	596,9	541,0
5.procent	593,0	627,7	361,0	395,6	378,3	388,7	873,5	352,5	442,1	375,6
10.procent	384,7	588,2	429,3	628,4	742,9	365,6	463,3	708,2	488,1	398,1
15.procent	355,1	364,3	341,3	478,6	476,8	400,7	566,0	432,3	559,8	1007,3
20.procent	546,9	448,9	413,2	588,1	389,7	453,1	520,2	433,7	1530,6	393,5
25.procent	365,6	466,3	391,5	359,1	372,1	319,6	493,0	363,8	1079,3	446,7
30.procent	471,9	472,0	388,8	615,4	420,9	479,6	1481,2	531,9	951,3	453,8
35.procent	614,2	502,6	722,2	761,8	1001,6	1765,0	491,9	2016,6	1854,0	605,7

Iterace SDP		
	Průměr	Nejlepší
0.procent	2004	2004
5.procent	2013,7	1999
10.procent	2061,9	2022
15.procent	2060,3	1971
20.procent	2430,4	2006
25.procent	2211,1	1979
30.procent	2670,7	2081
35.procent	3527,9	2100

Iterace MLAO*		
	Průměr	Nejlepší
0.procent	20099	20099
5.procent	20401,9	20099
10.procent	20976,6	20099
15.procent	21629,4	20099
20.procent	24325,1	20909
25.procent	22563,2	20099
30.procent	28309,3	22154
35.procent	39791,4	23004

ČasSDP[ms]		
	Průměr	Nejlepší
0.procent	4,091	3,902
5.procent	4,173	3,468
10.procent	9,173	3,824
15.procent	4,922	4,162
20.procent	6,360	4,463
25.procent	6,168	4,411
30.procent	5,628	4,204
35.procent	10,130	5,122

Čas MLAO*[ms]		
	Průměr	Nejlepší
0.procent	509,470	323,164
5.procent	478,796	352,462
10.procent	519,670	365,603
15.procent	498,221	341,302
20.procent	571,787	389,653
25.procent	465,704	319,621
30.procent	626,703	388,845
35.procent	1033,568	491,948

TEST 4.

V tomto testu byl startovní stav v poloze x:0, y:0 a cílový v 200,200. Optimální cesta by tedy měla prostorem probíhat diagonánálně. Z výslednků testů je patrno markantní zvyšování potřebného času a iterací pro výpočet MLAO*.

Iterace SDP										
	1	2	3	4	5	6	7	8	9	10
0.procent	2680	2680	2680	2680	2680	2680	2680	2680	2680	2680
5.procent	2750	2698	2670	2760	2755	2660	2720	2693	2737	2665
10.procent	2756	2757	2784	2755	2718	3090	2719	2759	2838	2810
15.procent	3071	3105	2899	2842	2896	2746	2919	3151	2983	3026
20.procent	2746	2963	3157	2879	3241	2750	2761	4170	2934	3871
25.procent	3603	3172	3091	3505	2915	6605	3624	3612	3761	3073
30.procent	4105	3954	3457	3355	7092	3237	4094	3083	3184	3372
35.procent	4791	3317	4079	3133	3829	6961	4088	3750	4288	4049

Iterace MLAO*										
	1	2	3	4	5	6	7	8	9	10
0.procent	20099	20099	20099	20099	20099	20099	20099	20099	20099	20099
5.procent	20705	21944	22154	21527	21735	22577	21320	20705	21320	21944
10.procent	24309	21114	23004	21320	24089	24752	30627	23219	24752	21114
15.procent	27729	28679	39620	27260	25650	24089	26564	32639	25199	24975
20.procent	31374	28679	29402	39339	54284	46359	31625	51359	35510	42485
25.procent	35244	30380	39620	28440	43955	70499	40754	29645	86319	27027
30.procent	97460	46664	46055	39620	45450	44550	44252	30627	55277	38502
35.procent	58995	43364	75465	87570	60725	48204	68264	248864	87989	49769

Čas SDP[ms]										
	1	2	3	4	5	6	7	8	9	10
0.procent	9,8	9,3	8,5	9,3	8,3	8,3	9,3	9,3	9,0	9,3
5.procent	9,4	8,5	8,5	10,2	9,2	8,8	9,7	9,6	8,6	9,6
10.procent	9,2	8,4	10,0	8,6	9,6	11,0	8,8	8,9	9,3	14,3
15.procent	11,4	10,9	10,7	11,0	10,0	11,2	11,3	11,3	11,0	10,0
20.procent	9,0	9,2	9,2	8,2	9,9	13,2	7,5	9,5	8,2	10,5
25.procent	9,9	10,1	8,5	9,4	8,1	15,6	9,0	9,4	9,9	8,4
30.procent	10,7	9,0	9,4	10,2	16,2	8,9	10,1	8,3	12,9	8,6
35.procent	15,2	8,7	10,0	8,7	10,5	13,2	10,7	9,5	9,4	10,9

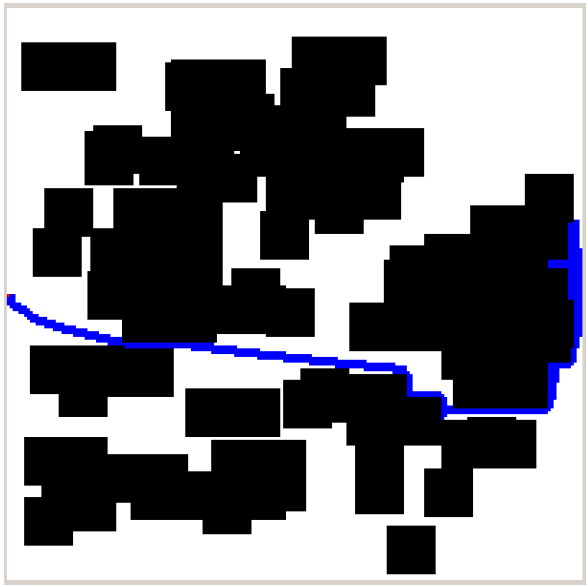
Čas MLAO*[ms]										
	1	2	3	4	5	6	7	8	9	10
0.procent	374,0	374,0	391,6	392,5	380,7	380,6	392,5	386,3	379,9	392,1
5.procent	398,7	606,0	602,0	533,2	539,4	439,3	422,2	397,0	409,6	416,2
10.procent	486,7	396,2	552,6	413,7	500,3	518,4	888,6	619,4	805,8	434,5
15.procent	586,3	719,0	1206,1	587,0	612,3	996,3	1573,5	943,6	625,3	647,7
20.procent	664,0	595,4	620,3	1665,5	1656,3	1280,5	791,3	2121,8	800,4	1154,0
25.procent	918,3	685,5	1448,7	569,6	1136,1	2439,3	1362,1	758,9	3489,1	601,5
30.procent	5159,0	1245,3	1731,9	1255,6	1288,2	1193,5	1781,0	623,7	1502,0	985,1
35.procent	2408,2	1190,3	2599,1	4189,8	1754,0	1211,7	2645,0	16330,3	3233,0	1988,9

Iterace SDP		
	Průměr	Nejlepší
0.procent	2680	2680
5.procent	2710,8	2660
10.procent	2798,6	2718
15.procent	2963,8	2746
20.procent	3147,2	2746
25.procent	3696,1	2915
30.procent	3893,3	3083
35.procent	4228,5	3133

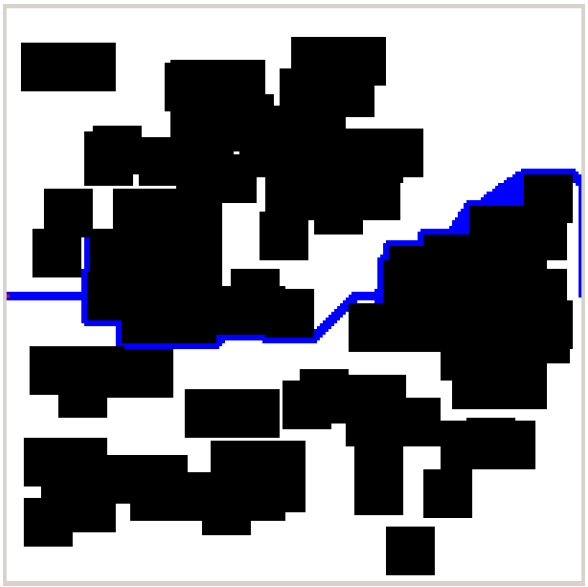
Iterace MLAO*		
	Průměr	Nejlepší
0.procent	20099	20099
5.procent	21593,1	20705
10.procent	23830	21114
15.procent	28240,4	24089
20.procent	39041,6	28679
25.procent	43188,3	27027
30.procent	48845,7	30627
35.procent	82920,9	43364

Čas SDP[ms]		
	Průměr	Nejlepší
0.procent	9,039	8,251
5.procent	9,208	8,464
10.procent	9,804	8,418
15.procent	10,883	10,016
20.procent	9,433	7,458
25.procent	9,839	8,091
30.procent	10,425	8,270
35.procent	10,691	8,685

Čas MLAO*[ms]		
	Průměr	Nejlepší
0.procent	384,407	373,977
5.procent	476,365	397,004
10.procent	561,618	396,162
15.procent	849,725	586,320
20.procent	1134,936	595,361
25.procent	1340,901	569,610
30.procent	1676,539	623,694
35.procent	3755,031	1190,328



Obr. 25 Příklad procházení prostředím č.2. Algoritmus SDP nahoře, dole pak modifikovaný LAO. Prostředí č.2 vytváří shluky překážek a tím i vyšší výpočetní náročnost plánování trasy.*



6 ZÁVĚR

V práci jsem se zabýval řešením plánování cesty robota pomocí dynamického programování. Jako model prostředí pro aplikaci algoritmů jsem zvolil mřížku. Zpracoval jsem algoritmus směrovaného dynamického programování a mé vlastní modifikace algoritmu LAO*. Úpravou algoritmu LAO* jsem chtěl docílit zkrácení výpočetního času a tím i lepšího srovnání s algoritmem SDP. Oba tyto algoritmy využívají jako základ heuristické metody plánování a jako nadstavbu a prvek umožňující efektivnější výpočty využívají metod dynamického programování. Algoritmy byly testovány postupně v generovaném prostředí, ve kterém se postupně zvyšoval procentuální podíl překážek. Bylo měřeno, kolik je zapotřebí aktualizací stavů a kolik času ten který algoritmus potřebuje k naplánování optimální trasy.

Jako výhodnější algoritmus lze podle výsledků testů označit algoritmus směrovaného dynamického programování. Byla otestována a ověřena jeho úspěšnost a efektivita při hledání cesty, zejména jeho menší výpočetní složitost a tím i menší potřeba času k nalezení optimálního řešení. Rozdíl ve výpočetní složitosti se projevil již při zvýšeném počtu překážek v prostoru, nebo aplikací těchto algoritmů na náhodné prostředí vytvářející v prostředí shluky překážek. Důvodem vyšší efektivity algoritmu SDP je skutečnost, že neprovádí aktualizaci hodnoty pro celý prostor již prozkoumaných stavů nýbrž pouze na ty jež jsou součástí okolí stavu vyjmutého z prioritní fronty a stav vyjmutý z fronty samotný. Počet aktualizací je tudíž omezený. Úprava algoritmu MLAO* se tedy neosvědčila.

Oba algoritmy prokázaly schopnost naplánovat optimální trasu v různých prostředích s různým počtem a velikostí překážek. Algoritmus SDP lze vzhledem k jeho efektivitě užít v praxi.

7 SEZNAM POUŽITÉ LITERATURY

[1]Bessiere P., Ahuactzin J.M.,Talbi E.G., Mazer E.: The "Ariadne's clew" algorithm:Global planning with local methods,J. Artificial Intell. Res ,9-295,9.,1996,

[2]LaValle S.M.: Planning Algorithms,Cabridge University Press , 2006,http://planning.cs.uiuc.edu/0521862051

[3]Lavrence A.,Hierarchical robot path planning using a distributed blackboard,1990,http://dspace.rice.edu/handle/1911/13422

[4]Ferguson D., Likhachev M., Stentz A.: A Guide to Heuristic-based Path Planning, 2005, International Conference on Automated Planning and Scheduling (ICAPS),,,http://www.pittsburgh.intel-research.net/~dfergus1/downloads/overview.pdf

[5]Shlomo Z.: Using Anytime Algorithms in Intelligent Systems,Intelligence ,1402-1407,,1993,http://anytime.cs.umass.edu/shlomo/papers/aimag96.pdf

[6]Latombe J. C.: Robot motion planning,Springer , 1991,0792391292

[7]Warren C. W.: Global Path Planning Using Artifical Potential Fields, 1989, IEEE International Conference,316-321,0-8186-1938-4,

[8]Ashiru I., Czarnecki Ch., Routen T.: Characteristics of a genetic based approach to path planning for mobile robots,Journal of Network and Computer Applications ,149-169,19(2),1996,

[9]Sedighi, K.H., Ashenayi, K., Manikas, T.W., Wainwright, R.L: Autonomous robot navigation using a genetic algorithm with an efficient genotype structure,Intelligent Engineering Systems Through Artificial Neural Networks: Smart Engineering Systems Design: Neural Networks, Fuzzy Logic, Evolutionary Programming, Complex Systems and Artificial Life ,319-324,,2004,http://www.ee.utulsa.edu/~tmanikas/Pubs/Hermanu-ANNIE-04-final.pdf

[10]Tarokh M.: A Genetic Robot Path Planner with Fuzzy Logic Adaptation, 2007, Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference,388-393,0-7695-2841-4,

[11]Ritthipravat P., Nakayama K.: Obstacle Avoidance by Using Modified Hopfield Neural Network, 2002, Proceedings of the International Conference on Artificial Intelligence,,,http://dspace.lib.kanazawa-u.ac.jp/dspace/bitstream/2297/11917/1/TE-PR-NAKAYAMA-K-161.pdf

[12]Bugmann G., Taylor J.G., Denham M.J.: Route Finding by Neural Nets,Neutral networks ,217-230,,1995,http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.105.3156&rep=rep1&type=pdf

[13]Yang S.X., Meng M.: Neural network approaches to dynamic collision-free trajectory generation,Systems, Man, and Cybernetics, Part B: Cybernetics ,302 - 318,31(3),2001,

[14]Sutton R. S., Barto A. G.: Reinforcement Learning:An Introduction,MIT Press , 1998,http://www.cs.ualberta.ca/~sutton/book/the-book.html0-262-19398-1

[15]Klapka J.,Dvořák J.,Popela P.: Metody operačního výzkumu,Vutium , 2001,80-214-1839-7

[16]Dean T., Kaelbling L.P.,: Planning under time constraints in stochastic domains,Artificial Intelligence ,35-74,76.,1993,http://www.sciencedirect.com/science?_ob=MImg&_imagekey=B6TYF-409W49M-3-1&_cdi=5617&_user=10&_orig=search&_coverDate=07%2F31%2F1995&_sk=999239998&view=c&wchp=dGLbVzW-zSkzS&md5=931c5bff2c58e918f484d32514fda9f5&ie=/sdarticle.pdf

[17]Ferguson, D., Stentz, A.: Focussed Processing of MDP's Path Planning, 2004, ICTAI 2004. 16th IEEE International Conference,310-317,0-7695-2236-X,

[18]Hansen E.A., Zilberstein S.: LAO*: A heuristic search algorithmthat finds solutions with loops,Artificial Intelligence ,35-62,129.,2001,
[19]Zhuang H., Du S., Wu T.: On-line real-time path planning of mobile robots in dynamic uncertain environment,Journal of Zhejiang University SCIENCE A ,516-524,7(4),2006,
[20]Moore A. W., Atkeson Ch. G.: Prioritized Sweeping: Reinforcement Learning WithLess Data and Less Time, ,,,1993,http://www.cs.jhu.edu/~hager/Teaching/cs336/Notes/Chap4-Potential-Fields.pdf
[21]http://www.cs.jhu.edu/~hager/Teaching/cs336/Notes/Chap4-Potential-Fields.pdf
[22]http://gamma.cs.unc.edu/COMP290-58/SLIDES/Lecture2.pdf&ei=EfcbSqHaLI-xsAaV9-CQAg&sa=X&oi=spellmeleon_result&resnum=1&ct=result&usg=AFQjCNHHShvswU0X_v7D7CdqbbZ3oyqyrQ
[23]http://www.es.aau.dk/fileadmin/es-files/projects/robotics/course/Kumar_-_Cell_Decomposition.pdf