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 ROBOTU POMOCÍ ALGORITMŮ AO*

ROBOT PATH PLANNING BY MEANS OF ALGORITHMS AO*

DIPLOMOVÁ PRÁCE
DIPLOMA THESIS

AUTOR PRÁCE **Bc. Tomáš Richter**
AUTHOR

VEDOUCÍ PRÁCE **RNDr. Jiří Dvořák, CSc.**
SUPERVISOR

BRNO 2015

ZADÁNÍ ZÁVĚREČNÉ PRÁCE

ABSTRAKT

Diplomová práce analyzuje metody plánování cesty robota v neurčitém prostředí pomocí metod typu AO*. Praktická část práce se zaměřuje na implementaci vybraných metod AO*. Pro provedení experimentů bylo vytvořeno simulační prostředí, které umožňuje otestovat implementované metody.

ABSTRACT

This diploma´s thesis analyzes methods for robot path planning by means of algorithms AO*. The practical part focuses on the implementation of selected methods AO*, which are designed for planning under uncertainty environment. There was created the simulation program in this work. Simulation program enables testing the methods, that were implemented.

KLÍČOVÁ SLOVA

Plánování cesty robota, neurčité prostředí, metody AO*

KEYWORDS

Robot path planning, uncertain environment, AO* methods

PROHLÁŠENÍ O ORIGINALITĚ

Prohlašuji, že jsem diplomovou práci vypracoval samostatně s využitím uvedených pramenů a literatury.

V Brně dne 29.5. 2015 …….

Podpis

BIBLIOGRAFICKÁ CITACE

RICHTER, T. *Plánování cesty robotu pomocí algoritmů AO**. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2015. 74 s. Vedoucí diplomové práce RNDr. Jiří Dvořák, CSc..

PODĚKOVÁNÍ

Zde bych chtěl poděkovat vedoucímu práce RNDr. Jiřímu Dvořákovi, CSc. za konzultace, cenné rady a podměty, které přispěly k napsání a realizaci této práce.

Obsah:

1 ÚVOD

Plánování cesty je v současné době aktuální téma. Dobré naplánování cesty uspoří náklady, které se musí na uskutečnění dané cesty vynaložit. V případě automobilu se jedná o palivo. Dopravní firmy mohou díky dobrému plánování doručit více zásilek, případně mohou ušetřit také na palivu. V případě robota nebo i člověka dochází při dobrém naplánování také k úsporám energie, která se musí vynaložit na absolvování cesty. Klíčové je plánování cesty u robotů, kteří jsou vysílání na vzdálené planety. Úkolem robota na vzdálené planetě může být například pořízení snímků určitých míst. V případě, že plán nebude vycházet, bude robot nucen vytvářet plán nový. Časté selhávání plánu by mohlo mít velmi negativní dopady na cíl mise. Na selhání vytvořeného plánu mohou mít vliv události, které mají neurčitý charakter. Robot může například špatně vyhodnotit data ze snímačů. Chybné vyhodnocení dat ze snímačů může zapříčinit chybnou identifikaci průchodnosti daného místa. Neprůchodnost robot zjistí až po přiblížení se k danému místu. Do plánu cesty je vhodné zahrnout neurčitosti prostředí. Zahrnutí neurčitostí prostředí do plánu cesty umožní předpovídat neurčitosti senzorů i akčních členů robota. Plánování cesty robota v neurčitém prostředí nám otevírá nové možnosti využití. Robotika proniká do stále více odvětví lidské práce, kde usnadňuje člověku práci i zachraňuje životy. Jako příklad je možno uvést rover při pracích na měsíci, úklidové roboty či roboty pracující v sutinách po zemětřesení. Tyto všechny využívají metod plánování své cesty.

Tématika plánování cesty robota v neurčitém prostředí pomocí metod typu AO* je v českých kruzích prozatím málo diskutovaným tématem. Práce s řešením plánování cest v takovém prostředí můžeme nalézt především u našich zahraničních kolegů, díky kterým se dostáváme pod povrch této oblasti. Díky využití nejnovějších poznatků jsme schopni nejen naplánovat trasu, ale dokážeme také předvídat neurčitosti té trasy s ohledem na optimální výsledek.

Téma práce jsem zvolil pro jeho zajímavost a neotřelost. Předpokládal jsem, že díky němu budu moci své dovednosti rozšířit v další oblasti, což se v průběhu psaní práce potvrdilo. Cílem této práce je provést analýzu metod plánování cesty robota v neurčitém prostředí, se zaměřením především na metody typu AO*. Praktická část práce se zaměří na implementaci vybraných metod plánování cesty robota. Zapotřebí bude taktéž vytvořit simulační prostředí, které umožní otestování implementovaných algoritmů.

Práce je tvořena teoretickou (kapitoly 2 a 3) a praktickou částí (kapitoly 4 až 6). Ve druhé kapitole popisuji plánování cesty robota jako takové pro celkový vhled do tématu. Ve třetí kapitole popisuji plánování cesty robota v neurčitém prostředí. Zde pojednávám o nástrojích a metodách, které umožňují plánování cesty. Čtvrtá kapitola se zabývá popisem simulačního prostředí, které jsem v rámci této práce vytvořil. Pátá kapitola řeší implementaci vybraných algoritmů do simulačního prostředí. Šestá kapitola pojednává o porovnávacích experimentech v simulačním prostředí. Pro práci jsem využil literárních zdrojů především z oblasti robotiky a optimalizace.

2 PLÁNOVÁNÍ CESTY ROBOTA

Využití metod plánování cest robota je velké množství. Uplatnění nacházejí od již zmiňovaného roveru, přes roboty, kteří pomáhají v rizikových oblastech zemětřesení či vulkanické činnosti až po nemocniční prostředí a úklidové práce.

Plánování cesty robota začíná tím, že robot dostane mapu prostředí s vyznačenými GPS souřadnicemi startu a cíle cesty. K tomu aby byl robot schopen s mapou pracovat se využívá různých metod, které můžeme dělit na online a offline metody. Offline metody jsou metody, které naplánují cestu před vyjetím robota. Tyto metody jsou využitelná při dobré znalosti prostředí, které se nemění. Online metody jsou využívány při plánování cesty ve stochastickém i deterministickém prostředí. Robot je díky nim schopen reagovat na nastalé situace a řešit je.

Dalším hlediskem pro plánování je typ prostředí. Prostředí můžeme dělit na stochastické a deterministické a dále na statické a dynamické. Stochastické prostředí je takové, které uvažuje plánování cesty robota s neurčitostmi jako jsou různé povrchy, nízké rozlišení mapy prostředí, neurčitosti senzorů a akčních členů robota, apod. Deterministické prostředí je prostředím, kde nejsou nejistoty neboli je nám dobře známé. Statické prostředí uvažuje stálé překážky neměnící svoji pozici (domy, kopce, vodní plochy) na rozdíl od prostředí dynamického, kde je předpokládáno, že se překážky mohou měnit (vozidla na silnici, pohyb osob či zvířat, přírodní úkazy).

Problém plánování cesty se široce řeší v oblasti robotiky. Existuje mnoho algoritmů pro řešení plánování cesty v deterministickém prostředí (například velmi známý algoritmus A* a jeho modifikace). V reálném prostředí však není zcela jisté, že naplánovaná akce bude provedena. V reálných podmínkách se robot setkává s řadou neurčitostí, které mohou zásadně, většinou negativně ovlivnit provedení plánu cesty. Reálné prostředí se mění. Je ovlivněno ročními obdobími, povětrnostními vlivy a dalšími faktory. Prostředí, pro které je cesta plánována, obsahuje mnoho různých povrchů (listí, tráva, led, …). Může také nastat situace, kdy robot špatně lokalizuje svoji pozici. Tyto proměnné mohou mít špatný vliv na pohyb robota a následně tedy i na splnění plánu cesty. Všechny naplánované akce se robotovi nemusí podařit vykonat vlivem neurčitostí prostředí. Pro vytvoření plánu cesty pro reálné podmínky je zapotřebí se zaměřit na algoritmy, které dokáží pracovat s neurčitostmi.

2.1 Stavový prostor a plánování cesty

Mezi široce využívané metody řešení úloh umělé inteligence patří prohledávání stavového prostoru. Prohledávání stavového prostoru je definováno množinou stavů, množinou akcí (operátorů), které umožňují přechod mezi stavy, počátečním stavem a množinou cílových stavů (Hansen, 2001). Stavy u deterministického rozhodovacího problému jsou v jakékoliv chvíli plně pozorovatelné. U plánování cesty robota vyjadřují jeho pozici v prostředí (Ferguson, 2004). Cílem je najít sled akcí, které zajistí přechod z počátečního stavu do cílového s účelem minimalizovat cenu dané cesty. Problém plánování cesty se převádí na problém prohledávání stavového prostoru (Hansen, 2001).

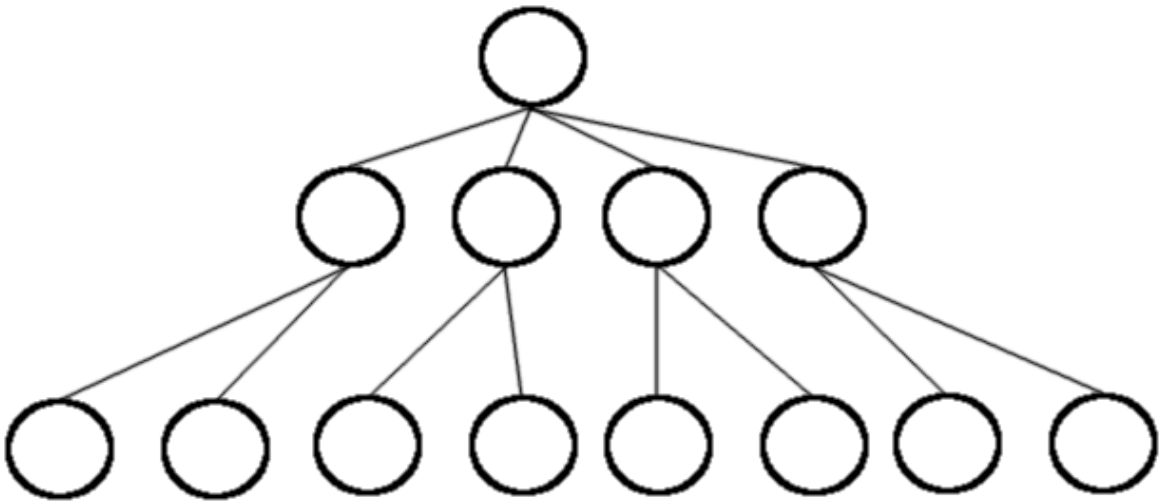
Formálně lze deterministickou stavovou reprezentaci úloh definovat dle (Bonet, 2005) následovně:

- diskrétním a konečným stavovým prostorem S,
- počátečním stavem $s_0 \in S$,
- neprázdnou množinou koncových stavů $S_T \subseteq S$,

- akcemi $A(s) \subseteq A$, které jsou aplikovatelné v každém neterminálním stavu $s \in S$,
- stavovou přechodovou funkcí f, která utvoří množinu stavů $F(s,a)$ pro $s \in S$ a $a \in A(s)$,
- cenou akce $c(a,s)$ pro každý neterminální stav s,
- cenou akcí v terminálních stavech $c_T(a,s)$,
- případně lze cenu přechodu ze stavu do stavu značit též $c(s,s')$, $s \in S$, $s' \in S$.

U uvedené stavové reprezentace se předpokládá, že $A(s)$ a $F(s,a)$ jsou neprázdné množiny, ceny jednotlivých akcí $c(a,s)$ jsou kladné, a terminální ceny $c_T(a,s)$ jsou nezáporné. Pokud se terminální cena rovná nule, pak se terminální stavy nazývají cílové. Stochastický model stavové reprezentace může vzniknou z deterministického modelu s využitím AND/OR grafů. (Bonet, 2005) Příklady plánování cesty v deterministickém prostředí, na základě zmíněného modelu, lze nalézt například v LaValle (2006).

Stavový prostor se obvykle popisuje pomocí grafů (Popelka, 2015). Jedná se o grafy známé z teorie grafů. Grafy jsou v tomto případě definovány množinou vrcholů a množinou hran. Dle Šedy (2006) je graf G dvojice (V, H), kde V je množina vrcholů grafu G a H je množina hran grafu G. Prostý graf má násobnost každé hrany rovnu nejvýše jedné. Základní pojmy z teorie grafů lze najít například v Šedovi (2006).



Obr. 1: Ilustrace neorientovaného grafu. Vrcholy grafu (stavy) jsou znázorněny kružnicemi. Hrany grafu (přechody mezi vrcholy, či mezi stavy ve stavovém prostoru) představují spojnice mezi kružnicemi v obrázku.

3 PLÁNOVÁNÍ CESTY V NEURČITÉM PROSTŘEDÍ

Tato kapitola pojednává o pravděpodobnostních plánovacích problémech hledání nejkratší cesty ze startu do cíle. Jedná se o rozšíření deterministického rozhodovacího problému. Klasické algoritmy určené k hledání optimální cesty ze startu do cíle v určitém prostředí (např. A*) nejsou schopné najít nejlepší řešení v neurčitém prostředí. Je důležité uvažovat neurčitosti prostředí pro naplánování cesty v reálném světě. (Ferguson, 2004) Na naplánované cestě se v reálném prostředí může vyskytnout překážka. Tato překážka může mít za následek neprůchodnost dané cesty (například sníh na cestě). (Aksakalli, 2007) Zjištění zda cesta je, či není průjezdná, může být učiněno až po přiblížení se k danému problematickému místu. Tento problém je v literatuře (Papadimitriou, 1991) popsán jako problém Kanadského cestujícího (originální název: „Canadian Traveller´s Problem“). V tomto případě tedy nejen, že existuje nejistota spojená s průchodností těchto cest, ale důležité jsou především následky v případě neprůchodnosti těchto cest. Následkem může být například o mnoho vyšší cena cesty v případě následného přeplánování (Ferguson, 2004). Pravděpodobnostní výskyt překážek je jeden typ neurčitostí, který se do stochastického modelu prostředí zavádí. Neurčité jsou však i důsledky jednotlivých akcí robota, jak je uvedeno v (Hansen, 2001). Přesnější a formální definice termínu akce bude uvedena až dále v textu. Prozatím pojem „akce“ představuje požadavek na přesun robota z pozice A na pozici B. V deterministickém prostředí požadovaná akce proběhne se 100%-ní pravděpodobností – tedy skutečně dojde k přesunu robota z pozice A do B. Ve stochastickém prostředí tomu tak být nemusí. Výsledkem požadavku na přesun z pozice A do B, nemusí být vždy B, ale může to být i pozice jiná (dokonce se akce nemusí provést – tedy výsledkem dané akce je opět pozice A). Za tuto skutečnost může například povrch, po kterém se robot pohybuje. Například led na cestě může způsobit proklouznutí kol, čímž místo k přesunu z bodu A do bodu B, dojde k přesunu z A do bodu C (Chung, 2011). Skutečné prostředí obsahuje mnoho různých typů neurčitostí. Některé možné zdroje neurčitostí jsou stručně shrnuty níže v bodech.

- Stochastický výskyt překážek – dynamické prostředí (Papadimitriou, 1991)
- Stochastické akce robota (Hansen, 2001)
- Kontinuální zdroje (včetně času) se stochastickou spotřebou (Benazera, 2005)
- Možné jednorázové odměny dosažitelné pouze v jednom tahu (Benazera, 2005)

V neurčitém prostředí je možné vytvořit plán cesty s využitím algoritmů AO*. Pro plánování cesty se využívají různé varianty algoritmu AO*, například BAO* (Aksakalli, 2007), LAO* (Hansen, 2001), PAO* (Ferguson, 2004) a další.

Neurčité prostředí je možné simulovat za pomoci AND/OR grafu. Metody AO* umí prohledat stavový prostor tvořený AND/OR grafem. Výhodou těchto metod je, že neprohledávají celý stavový prostor. Jedná se o heuristické metody prohledávání stavového prostoru. Metody AO* by oproti metodám, které prohledávají celý stavový prostor, měly dosahovat výsledného řešení mnohem rychleji.

3.1 AND/OR grafy

Jednou ze speciálních variant grafu je AND/OR graf. AND/OR graf lze dle Martelliho (1978) definovat jako hypergraf. U hypergrafu se vyskytují hyperhrany, které spojují n-tice vrcholů (n = 1, 2, 3, ...). Hyperhrany se nazývají konektory a jsou v případě

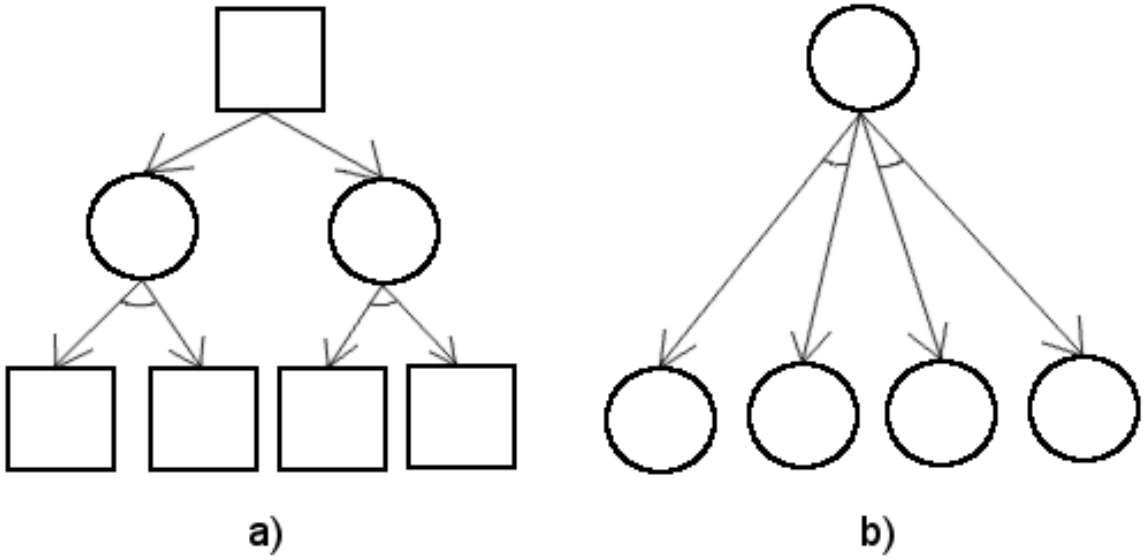
AND/OR grafu chápány jako orientované z jejich prvního vrcholu do všech ostatních. Formálně je AND/OR graf dvojice (V, K), kde V je množina vrcholů grafu a K je množina konektorů (1) (Martelli, 1978).

$$K \subseteq V \times \bigcup_{k=0}^{m} V^k \tag{1}$$

Každý k-konektor (v_{i_0}, v_{i_1}, v_{i_2}, …, v_{i_k}) je uspořádaná (k+1)-tice, kde v_{i_0} je vstupní vrchol a v_{i_1}, v_{i_2}, …, v_{i_k} jsou výstupní vrcholy. Vrchol v_{i_0} je rodič (nebo také předchůdce) vrcholů v_{i_1}, v_{i_2}, …, v_{i_k}, (potomků vrcholu v_{i_0}). Prostý graf je vytvořen za předpokladu $K \subseteq V^2$. V grafu se též může vyskytnout vrchol, která má 0-konektorů. Může to být například konektor s jedním vstupem a žádným výstupem (koncové vrchol). (Martelli, 1978)

AND/OR graf může být také definován jiným způsobem než pomocí konceptu hyperhran (k-konektorů). Martelli (1978) se odkazuje na standardní definici (Nilsson, 1971), která rozeznává AND a OR vrcholy. Dle této definice je AND/OR graf dvojice (V, H), kde V je množina vrcholů a H je množina hran. Množina vrcholů V se dělí na podmnožinu AND vrcholů značenou V_a, a podmnožinu OR vrcholů značenou V_o. U konceptu hyperhran (k-konektorů) jsou oproti standardní definici vrcholy AND nahrazeny konektory.

Obě definice (koncept AND a OR uzlů i koncept k-konektorů) jsou dle Martelliho (1978) navzájem ekvivalentní. Vztah obou konceptů je zobrazen na obrázku níže. (Obr. 2). Dle konvence uvedené v Hansenovi (2001) značí na obrázku (Obr. 2) čtverec OR vrchol a kružnice AND vrchol.



Obr. 2: a) Koncept AND a OR uzlů; b) koncept k-konektorů (Hansen, 2001)

Pro řešení problémů pomocí AND/OR grafu je zapotřebí zadefinovat řešitelnost jednotlivých vrcholů. Nelistový vrchol, který je typu AND je řešitelný, pouze pokud jsou všichni jeho bezprostřední potomci řešitelní. Nelistový vrchol typu OR je řešitelný, pokud alespoň jeden jeho bezprostřední potomek je řešitelný. (Březina, Dvořák, 2013a)

Důležité je zavést ohodnocení AND/OR grafu, které umožní heuristickým algoritmům jeho prohledání a tím nalezení řešení zadaného problému. Ke každé hraně jsou

přiřazeny funkce $d: H \to \mathbb{R}_{\geq 0}$ určující délku každé hrany, a funkce $p: H \to [0,1]$ přiřazující pravděpodobnost každé hraně. Všechny hrany vystupující z vrcholu typu OR mají vždy pravděpodobnost rovnu jedné. (Aksakalli, 2007)

Nechť $S(v)$ označuje následníka vrcholu $v \in V$ v AND/OR grafu. Funkce $g: V \to \mathbb{R}_{\geq 0}$ je konzistentní na kolekci vrcholů $C \subset V$ za předpokladu, že pro všechny $v \in C$ jsou splněny následující podmínky (2) až (4): (Aksakalli, 2007)

$$\text{Pokud } v \in V_A,\ g(v) = \sum_{v' \in S(v)} [\, p(v,v') \cdot (d(v,v') + g(v'))] \tag{2}$$

$$\text{Pokud } v \in V_O,\ g(v) = min_{v' \in S(v)} \{ d(v,v') + g(v') \} \tag{3}$$

$$\text{Pokud } v \in V \text{ je listový vrchol, pak } g(v) = 0. \tag{4}$$

Funkce $f: V \to \mathbb{R}_{\geq 0}$, která je konzistentní na V se nazývá nákladová (váhová) funkce a $f(v)$ určuje pravdivé ohodnocení vrcholu v. Obvykle jsou hodnoty p a d zadány explicitně, a f je definováno implicitně skrze hodnoty p a d (Aksakalli, 2007).

AND/OR graf může reprezentovat rozklad úlohy na podúlohy. Využitelný je však také při plánování v neurčitém prostředí. Vrcholy typu OR představují stavy robota (např. jeho pozici). Možné akce robota jsou reprezentovány vrcholy typu AND. Ve vrcholu typu OR je možné zvolit jakoukoliv akci. Akce (vrcholy typu AND) musejí však být prozkoumány všechny (musí se zjistit všechny možné důsledky dané akce). Z vrcholů typu OR (stavů robota) tedy vedou hrany do vrcholů typu AND (akcí) a ty vedou následně do vrcholů typu OR (stavů příslušejících daným akcím) (Hansen, 2001).

U konceptu *k*-konektorů jednotlivé vrcholy reprezentují stavy robota. Hyperhrany (*k*-konektory) jsou interpretovány jako akce s neurčitým výsledkem. Akce může transformovat stav do jednoho *k* možných potomků stavu s určitou pravděpodobností přiřazenou ke každému potomkovi (Hansen, 2001).

AND/OR grafy lze využít při plánování cesty v neurčitém prostředí. Pro nalezení optimální cesty v neurčitém prostředí lze využit algoritmus AO* a jeho modifikace. Algoritmus AO* je určen k heuristickému prohledání ohodnoceného AND/OR grafu. Pro nalezení optimálního rozhodnutí využívá ohodnocení vrcholů pomocí nákladové funkce $f(v)$ (Hansen, 2001; Ferguson, 2004; Aksakalli, 2007).

3.2 Algoritmus AO*

Algoritmus AO* je heuristická metoda určená k prohledávání ohodnocených AND/OR grafů. AO* prohledává AND/OR graf postupným vytvářením grafu řešení z počátečního vrcholu pomocí dvou navzájem se střídajících fází. Nejdříve se rozšíří nejlepší částečné řešení expanzí jednoho neterminálního listového vrcholu a přiřadí se přípustné heuristické hodnoty (ceny) jeho potomkům. Následně se použijí nově vypočítané ceny k rozšíření revizí hodnot v rodičovském uzlu a jeho předchůdcích. (Ferguson, 2004) Graf řešení má podobu AND/OR stromu (Aksakalli, 2007). Jedná se o acyklický podgraf kompletního AND/OR grafu. (Hansen, 2001) Teoreticky by bylo dle (Aksakalli, 2007) možné určit optimální řešení problému zadaného AND/OR grafem, vypočítáním $f(v)$ pro všechny vrcholy $v \in V$. Vzhledem k exponenciálnímu růstu počtu vrcholů v AND/OR grafu by bylo určení optimálního řešení pomocí tohoto postupu velmi nákladné na výpočetní výkon. Na druhou stranu, pro určení optimálního řešení nemusejí být všechny vrcholy vyhodnoceny. Do prohledávání AND/OR grafu lze zanést heuristiky, které určí

jaké vrcholy je nutné vyhodnotit pro určení optimálního řešení. Klasický AO* algoritmus pro prohledávání AND/OR grafů vylepšuje metodu hrubé síly přípustnými odhady $h : V \rightarrow \mathbb{R}_{\geq 0}$, které se nazývají heuristickými značkami (heuristic labels). Tyto heuristické značky jsou dolními mezemi, které zaručí, aby nedošlo k nadhodnocování žádného vrcholu. Díky dolním mezím je prohledána pouze malá část kompletního AND/OR grafu (Aksakalli, 2007).

1) Počáteční graf *G* se skládá výhradně z počátečního vrcholu (vrchol INIT). Pro vrchol INIT je nutné vypočítat heuristický odhad *h*(INIT).
2) Pokud vrchol INIT není označen jako SOLVED nebo jeho hodnota heuristického odhadu *h*(INIT) je menší než FUTILITY (maximální povolená cena řešení), pak pokračuj v následujících pod-krocích.
   I. Procházej grafem *G* z vrcholu INIT a následuj označené hrany [nejnadějnější podstromy grafu *G* (jejich kořeny mají nejmenší heuristickou značku)], dokud nenarazíš na neexpandovaný vrchol VERTEX.
   II. Expanduj VERTEX. Pokud VERTEX nelze expandovat, pak je tento vrchol neřešitelný, přiřaď tedy *h*(VERTEX) hodnotu FUTILITY. Pokud má VERTEX potomky, označ každého z nich jako SUCCESSOR. Pro každý vrchol SUCCESSOR, který není zároveň předek vrcholu VERTEX proveď následující pod-kroky:
      a) Přidej vrchol SUCCESSOR ke grafu *G*.
      b) Pokud SUCCESSOR představuje konečný konečný (cílový/elementární) vrchol, označ ho jako SOLVED a přiřaď jeho heuristickému odhadu *h*(SUCCESSOR) hodnotu nula.
      c) Pokud SUCCESSOR není konečný vrchol, vypočítej jeho heuristický odhad.
   III. Rozšiř nově objevené informace grafem *G* pomocí následujících pod-kroků:
      a) Vytvoř prázdný seznam *S*, který bude obsahovat vrcholy jejichž ohodnocení může být změněno. Vlož do seznamu *S* vrchol VERTEX.
      b) Vyber ze seznamu *S* vrchol, jehož žádný potomek z grafu *G* není současně v seznamu S (nejprve je nutné zpracovat nejhlubší vrcholy). Vybraný vrchol označ jako CURRENT a odstraň ho ze seznamu S.
      c) V případě, že je vrchol CURRENT řešitelný, označ jej jako SOLVED, přiřaď mu heuristickou hodnotu *h*(CURRENT) nula a pokračuj bodem e).
      d) Stanov novou hodnotu *h*(CURRENT) pomocí vztahů (2) až (3) a označ nejnadějnější podstrom vrcholu CURRENT v případě, že vrchol CURRENT je typu OR.
      e) Pokud byl vrchol CURRENT označen jako SOLVED, nebo byla pozměněna hodnota *h*(CURRENT), přidej do seznamu *S* rodiče vrcholu CURRENT.
      f) Pokud není seznam *S* prázdný, pokračuj bodem b). Jinak pokračuj bodem 2).
3) Vrať optimální graf řešení *G*.

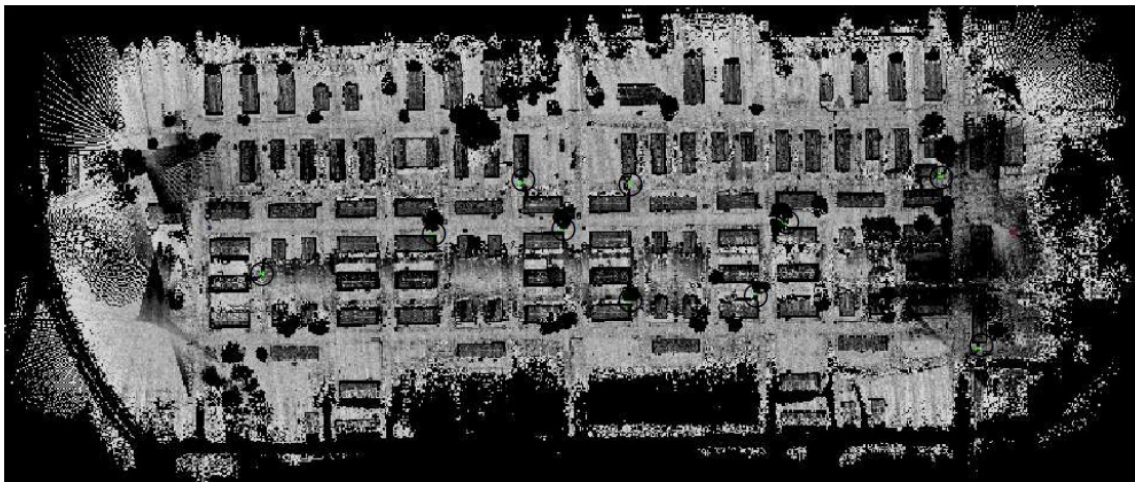
Algoritmus 1: AO (Březina, Dvořák, 2013a; Veera Raghavavaiah, 2010)*

3.3 Deterministické rozhodovací problémy se skrytými stavy

Navigovaný robot může být vybaven mapou okolního prostředí, ve kterém se pohybuje. Mapa prostředí může být získána například satelitem nebo bezpilotním letounem. Rozlišení takto získané mapy však nemusí být dostatečně vysoké pro bezproblémovou navigaci robota. Kvůli nízkému rozlišení mapy vzniká nejistota spojená s určitými místy na mapě. Z těchto určitých míst na mapě nemusí být zcela jasné, zdali se

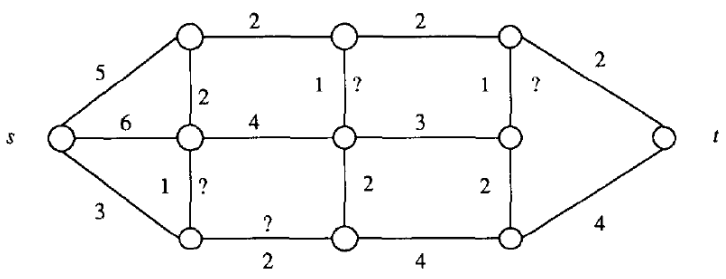
jedná o místo průchozí, či zda se jedná o překážku pro robota. Některé z těchto neurčitých míst na mapě mohou být stěžejní pro naplánování cesty. (Ferguson, 2004) Daný neurčitý bod může například ležet na nejkratší cestě z počátečního bodu do cílového. Pokud však robot naplánuje cestu skrze tento neurčitý bod a následně při provádění plánu se zjistí, že daný bod není průjezdný, může to mít vážné důsledky. Musí dojít k vytvoření nového plánu a prodlužuje se doba dojezdu ze startu do cíle. Na místo určení nemusí robot dorazit vlivem delší cesty včas. Případně mu také na samotnou cestu nemusí vystačit zdroje (jedná se pouze o příklad, tento model nebude uvažovat spotřebu zdrojů).

Příklad mapy, která by mohla být použita pro navigaci robota je na obrázku (Obr. 3). Jedná se o gradientní mapu venkovního prostředí získanou pomocí helikoptéry. Mapa zobrazuje prostředí o velikosti 300x700 metrů. Na mapě je znázorněno několik neurčitých bodů (zelené v černém kroužku). Z šedé škály barev lze vyčíst průchodnost jednotlivých bodů na mapě. Tmavší body znamenají obtížnější terén, pokud jsou body vybarveny černou barvou jedná se o překážku. (Ferguson, 2004)



Obr. 3: Gradientní mapa venkovního prostředí získané pomocí helikoptéry. (Ferguson, 2004)

Zavedením skrytých stavů do deterministického rozhodovacího problému se otevře možnost modelovat výše popsanou neurčitost. Zejména bude možné navigovat robota v prostředí, kde existují detekovatelné stavy. Ve vnitřním prostředí se mezi jednotlivými místnostmi mohou vyskytovat dveře – ty mohou být otevřené nebo zavřené. U venkovního prostředí se mohou vyskytnou cesty, které budou příliš úzké pro průchod robota. V navigovaném prostředí je také možný výskyt neprůchozích stavů, např. kvůli nízkému rozlišení mapy. (Ferguson, 2004) Popsaný problém také zahrnuje Problém Kanadského cestujícího („Canadian Traveller´s Problem“), který je známý z teorie grafů. Jedná se o problém průchodu grafovou strukturou, kde některé hrany mohou být neprůchozí (Papadimitriou, 1991).



Obr. 4: Problém Kanadského cestujícího (Papadimitriou, 1991) s – startovní bod, t – cílový bod

3.3.1 Skryté stavy

V prostoru, kde bude probíhat plánování se může vyskytovat p skrytých stavů. Každý tento bod v prostředí může být průchozí nebo neprůchozí. S možnou neprůchodností daného bodu mohou být spojeny negativní důsledky působící na celkový plán cesty. Např. výsledná cesta ze startu do cíle v případě přeplánování může být delší než kdyby se plánovací algoritmus danému bodu vyhnul, apod. S každým skrytým stavem je spojeno rozdělení pravděpodobnosti nad jeho možnými hodnotami. Skryté stavy budou v této práci brány jako statické. Hodnota, která se skrytému stavu přiřadí (průchozí, neprůchozí) bude neměnná (Ferguson, 2004).

Stav skrytého stavu může být zjištěn agentem. V tomto modelu skrytých stavů se nepředpokládá neurčitost akčních a senzorických členů. Agent je tedy schopen při objevování stavu s určitostí stanovit, zdali je tento stav průchozí nebo neprůchozí. Zároveň se nepředpokládá, že by akce mohla vést na více možných stavů (Ferguson, 2004). Model deterministického rozhodovacího problému se skrytými stavy, tak jak je definován v této práci, je příbuzný k částečně pozorovatelnému Markovovu rozhodovacímu procesu (Kaelbling, 1998). Výhodnou vlastností takto zadefinovaného modelu je možnost použití heuristických vyhledávacích algoritmů nad stavovým prostorem, který je částečně pozorovatelným Markovovým procesem neřešitelný (Ferguson, 2004).

3.3.2 Informační stavy

Informační stav obsahuje vědomosti agenta, které se týkají opravdového stavu prostředí. Informační stav obsahuje známé i skryté prvky stavového prostoru. Pro deterministický rozhodovací proces se skrytými stavy lze informační stav zapsat jako dvojici (s, I). Proměnná s značí pozorovatelný stav a $I = \{v(h_1), v(h_2), \ldots, v(h_p)\}$ je agentova znalost skrytých stavů. Každý skrytý stav může nabývat třech možných hodnot: $(v(h_i) = t)$ – průchozí stav; $(v(h_i) = o)$ – stav je překážkou; $(v(h_i) = u)$ – neznámý stav. Velikost plánovací mřížky známých stavů je dána vztahem $m \cdot n$. Při uvažování skrytých stavů je počet informačních stavů $m \cdot n \cdot 3^p$ (Ferguson, 2004).

Stavový prostor kvůli danému uvažování skrytých stavů roste exponenciálně. Plánování přes takový stavový prostor může být příliš nákladné (na výpočetní výkon apod.).

3.3.3 Graf sousedů

Všechny skryté stavy v prostředí mohou mít řadu sousedů, kteří jsou tvořeny přilehlými buňkami. Tyto sousedy si lze představit jako různé vchody a východy skrytého stavu. Každá buňka (jenž není překážkou) sousedící se skrytým stavem je právě jedním z jeho sousedů. Graf sousedů propojuje jednotlivé sousedy mezi sebou, čímž je zajištěna reprezentace skrytých stavů daného prostředí. Hrana mezi dvěma sousedy je hodnocena nejnižší možnou cenou cesty (bez skrytých stavů) mezi jednotlivými sousedy. Tímto způsobem se v literatuře problém prohledávání celého stavového prostoru (deterministického rozhodovacího problému se skrytými stavy) redukuje pouze na prohledávání skrytých stavů (Ferguson, 2004). Graf sousedů je možné z mapy prostředí vytvořit následujícím způsobem dle Fergusona (2004).

Za účelem vytvoření příslušných hran a jejich cen v grafu sousedů, je nutné šířit počáteční cenu algoritmem prioritního zametání (prioritized sweeping) přes prostředí. Algoritmus „prioritized sweeping“ popisuje například Moore a Atkeson (2012). Tento algoritmus určí pro každou buňku v plánovací mřížce cenu cesty do všech sousedů a cenu cesty do cíle. Při šíření počáteční ceny se se všemi skrytými stavy zachází jako s

překážkami. Tím se zjistí, kteří sousedé jsou navzájem dostupní. Poté se k vybranému sousedovi f_1 a odpovídajícím cenám přechodu z f_1 do všech další sousedů $c(f_1, f_2)$, $c(f_1, f_3)$, …, $c(f_1, f_n)$ vytvoří hrana do každého souseda f_i pro všechny $c(f_1, f_i) < \infty$ a označí se příslušnou cenou. Následně se vypočítá cena mezi sousedy, kteří náleží ke společnému skrytému stavu. Tato cena se využívá v informačních stavech, u kterých je pro skrytý stav přiřazena hodnota *t* – průchozí stav. Po kompletním vytvoření grafu sousedů se deterministický problém se skrytými stavy redukuje na úlohu Kanadského cestujícího.

3.3.4 Reprezentace a řešení úlohy pomocí AND/OR grafu

Úlohu Kanadského cestujícího lze reprezentovat pomocí AND/OR grafu. Řešení této úlohy lze najít prohledáním AND/OR grafu pomocí algoritmu AO*. Reprezentovat úlohu Kanadského cestujícího pomocí AND/OR grafu lze následovně. Každý uzel v grafu odpovídá sousedovi v částečném informačním stavu. Kořen grafu je uzel typu OR a reprezentuje počáteční buňku *s* (start) v informačním stavu $I = \{u, u, u, \ldots\}$. Další úroveň v grafu odpovídá všem sousedům, kteří jsou spojeni hranou s uzlem *s*. Všichni sousedé uzlu *s* jsou tedy jeho potomci. Každý tento potomek má informační stav *I*. Potomci uzlu *s* jsou uzly typu AND; každý tento uzel má dva potomky reprezentující dva možné informační stavy, které je možné dosáhnout z navštěvujícího uzlu. Tito dva potomci mají schodné sousedy jako jejich rodiče, ale sídlí v různých informačních stavech. Jednomu skrytému stavu je přiřazen soused s hodnotou *t* (traversable – průchozí) a druhému *o* (obstacle – překážka). Takto vzniklí potomci jsou typu OR. Další úroveň v grafu je typu AND, další OR, a tak dále (Ferguson, 2004).

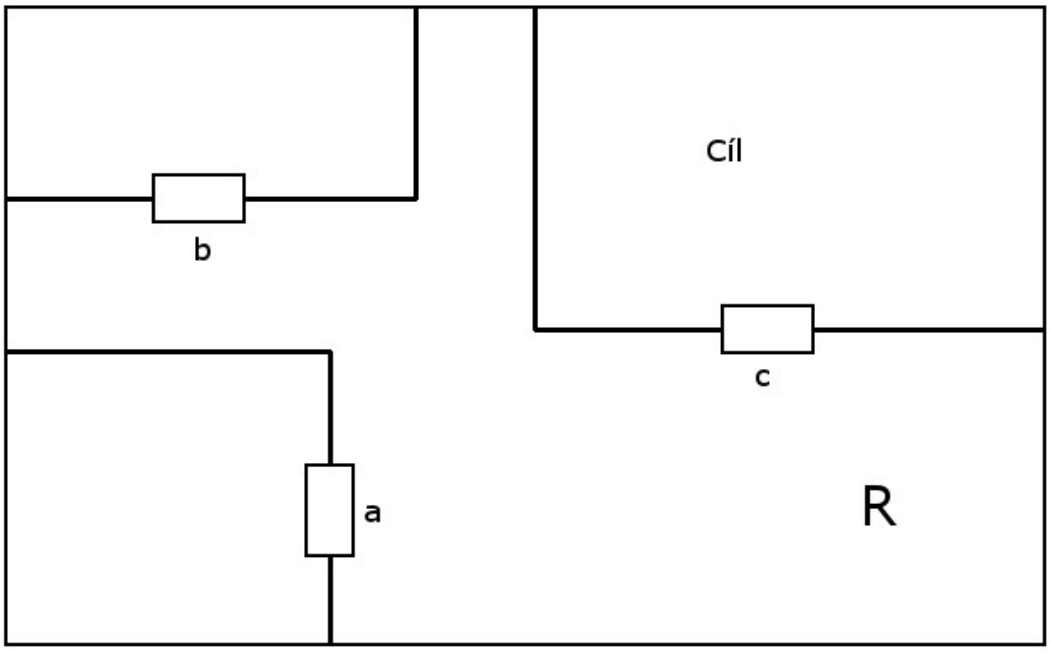
Agent se rozhoduje na volbě cesty na základě ocenění jednotlivých uzlů. Ocenění uzlů je dáno vztahy (2) až (3), které jsou uvedeny v podkapitole 3.1 AND/OR grafy. Optimální cestu je schopný nalézt algoritmus AO*. Přesný popis algoritmu AO* je uveden v podkapitole 3.2 Algoritmus AO*.

3.3.5 Algoritmus PAO*

Algoritmus AO* pracuje velmi dobře v jistých situacích, typicky pokud je většina z dosažitelných stavů jasně nežádoucí. Je to zapříčiněno tím, že použitá heuristika se dovoluje zaměřit na prohledávání, které se vyhýbá vysoce oceněným sousedům. Avšak vzhledem k danému problému (deterministické hledání nejkratší cesty se skrytými stavy) je možné, že AO* algoritmus bude prohledávat více stavů, než je potřeba. Kromě toho, graf řešení je během provádění algoritmu pozměňován, dochází k expandování jednoho a toho samého stavu několikrát. (Ferguson, Stentz, Thrun, 2004)

Zkratka algoritmu PAO* znamená v originálním znění – *Propagation* AO*. Algoritmus PAO* prohledává AND/OR graf postupným vytvářením grafu řešení z počátečního stavu pomocí dvou střídajících se fází, stejně jako klasický AO* (Ferguson, Stentz, 2004b). PAO* využívá výhody klasického AO* algoritmu s níž je spojeno heuristické vyhledávání. Navíc PAO* minimalizuje možné nevýhody, které jsou spojené s používáním podstromů grafu *G* (grafu kompletního řešení). Minimalizování nevýhod probíhá při zpětném šíření cen, kdy PAO* algoritmus nešíří změny pouze k rodičům v podstromu řešení, ale také do stran k sousedům (v kompletním AND/OR grafu). (Ferguson, Stentz, Thrun, 2004)

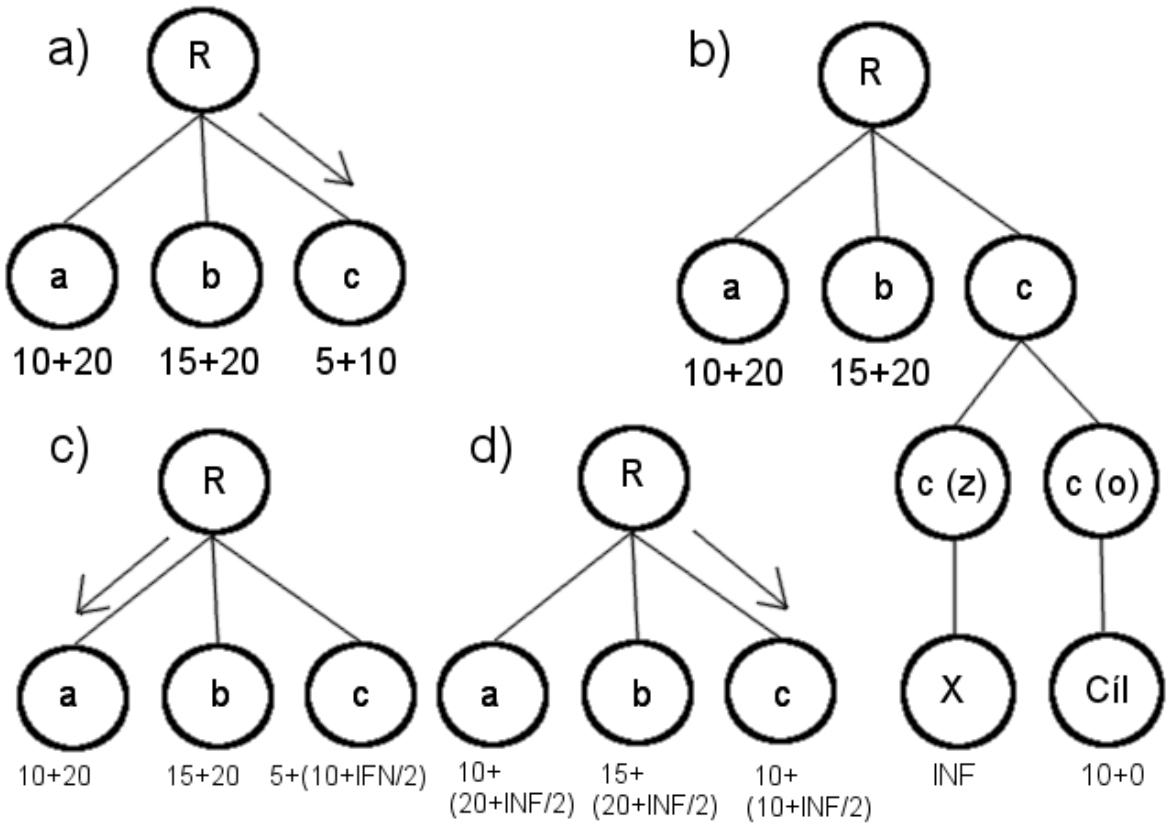
Před uvedením přesnějšího popisu algoritmu PAO* zde bude zmíněn jednoduchý příklad na stochastické hledání cesty, který vychází z literatury (Ferguson, Stentz, Thrun, 2004). Příklad předpokládá jednoduché prostředí, které je znázorněno na obrázku (Obr. 5). V tomto prostředí se nachází robot (označen písmenem R), jehož úkolem je dostat se z počáteční pozice do cílové. Robot bude procházet přes prostředí, kde se nacházejí dveře, které mohou být zamčené. Robot může po přiblížení se ke dveřím zjistit, zdali jsou otevřené nebo zavřené pomocí senzoru, který má k dispozici. Pro jednoduchost, robot se může z počáteční pozice přemístit pouze do tří možných pozic. K možným přesunům jsou v tabulce shrnuty jejich ceny. Příklad předpokládá, že dveře, které dělí robota od cíle jsou průchozí (*c*-(*o*) značení odemčené dveře, *c*-(*z*) dveře zamčené). Ceny přesunů robota na různé možné pozice jsou shrnuty v tabulce 1.



Obr. 5: Příklad plánovacího problému. Obrázek zahrnuje prostředí, výchozí polohu robota, cíl a tři dveře, které mohou být otevřené nebo zavřené. Ilustrováno dle (Ferguson, 2004).

Tabulka 1: Ceny hran pro příklad na Obr. 5

C(R,a) = 10	C(R,b) = 15	C(R,c) = 5
C(a,c) = 10	C(b,c) = 10	C(c (o),cíl) = 10



*Obr. 6 Příklad plánování cesty v neurčitém prostředí (příklad z obrázku (Obr.*5*))*

a) První část expanze kořenového uzlu (AO i PAO*). Pro druhou část expanze se vybere nejnadějnější vrchol.*

b) Druhá část expanze (AO i PAO*).*

c) Rozhodování algoritmu AO po propagaci cen*

d) Rozhodování algoritmu PAO po propagaci cen*

Upraveno podle (Ferguson, 2004)

Výše zmíněný příklad je dále rozveden na obrázku (Obr. 6). Jako nejlepší následovník je v první části expanze označen (s přihlédnutím k tabulce 1) vrchol *c*. Výpočet ohodnocení uzlů probíhá následovně (uvedeno na příkladu výpočtu ohodnocení pro uzel *c*, při přechodu z *R*): *Cena*(*R*) = *Cena*(*R*, *c*) + *Cena*(*c*, *cíl*) = 5 + 10. Vrchol *c* má dva potomky: jednoho pro případ, že dveře jsou otevřené a druhého pro případ, kdy jsou dveře neprůchozí. K těmto potomkům uzlu *c* je přiřazeno dané ohodnocení (pro neprůchozího potomka je ohodnocené rovno *INF* – nekonečno). Po ukončení expanze a následné propagaci ceny k potomkům se ohodnocení vrcholu *c* výrazně změní. Klasický algoritmus AO* v takovémto případě vrchol *c* nezvolí jako nejnadějnějšího (znázorněno na obrázku (Obr. 6e)) a pro další expanzi zvolí jiný vrchol – v tomto případě vrchol *a*. Jak uvádí Ferguson (2004), tímto způsobem nejsou efektivně využívány informace z předchozí expanze. Jelikož pro dosažení cíle je nutné navštívit vrchol *b*, ceny všech sousedů vrcholu *b* jsou ovlivněny změnami cen spojených i s vrcholem *b*. Algoritmus AO* expanduje

všechny sousedy, než vybere vrchol *b* jako nadějný pro nalezení cíle. Popsanou nevýhodu řeší algoritmus PAO*. PAO* šíří informace vztahující se k změnám cen více přes stavový prostor než algoritmus AO*. Stavový prostor je v takovém případě více informovaný. (Ferguson, Stentz, Thrun, 2004)

1. Počáteční graf *G* se skládá výhradně z počátečního vrcholu *s*, v původním informačním stavu *I*.
2. Pokud má graf *G* nějaké neterminální listové vrcholy, pak proveď následující podkroky:
   I. *Nalezni neexpandovaný neterminální vrchol*: Procházej grafem *G* dokud nenarazíš na neexpandovaný vrchol. Během průchodu grafu *G* aktualizuj cenu každého potomka, který je překážkou tak, aby jeho heuristické ohodnocení bylo zdola omezeno heuristickým ohodnocením rodičovského uzlu.
   II. *Expanduj nejnadějnější neexpandovaný neterminální vrchol*: Expanduj neterminální listový vrchol a vypočítej ceny jeho potomků. Potomkovi expandovaného vrcholu, který je průchozí (není překážkou), bude jeho cena rovna heuristickému ohodnocení. Potomci, kteří představují neprůchozí vrcholy (překážky) zdědí cenu od rodiče a následně je spuštěn algoritmus omezené iterace hodnoty (limited value iteration) přes jejich heuristické protějšky, aby se případně zvýšily zděděné ceny. Potomci se přidají ke grafu *G*, s případnou poznámkou, že jsou terminální.
   III. *Rozšíření změn cen a aktualizace grafu G*: Vypočítej a aktualizuj ceny původních listových vrcholů vzhledem k cenám jejich potomků. Pokud se cena vrcholu změnila, pak aktualizuj cenu odhadů pro jeho celý informační stav. Také je v případě změny ceny vrcholu zapotřebí aktualizovat cenu rodiče původního listového vrcholu. Pokud je rodič vrchol typu OR, pak aktuální vrchol může být nahrazen v případě, že dále neposkytuje minimální cenu. Pokud je vrchol průchozím potomkem, pak aktualizuj ceny spojené s celým rodičovským stavem tak, aby byly zdola omezeny aktuálním stavem. Pokračuj v šíření cen v grafu, dokud nebude dosažen vrchol, jehož cena se nezmění.
3. Vrať optimální graf řešení *G*.

Algoritmus 2: PAO (Ferguson, Stentz, Thrun, 2004)*

K tomuto popisu algoritmu (Algoritmus 2) je zapotřebí dodat poznámku k bodu 2) – II, kde je zmíněn termín heuristické protějšky (heuristic counterparts). Heuristický protějšek vrcholu *v* je dle Fergusona (2004) deterministický stav charakterizovaný nejlepšími možnými hodnotami skrytých prvků, které vrchol *v* může mít. Pro prvky, které jsou ve vrcholu *v* známé (například je-li $v(h) = t$ nebo $v(h) = o$ – dle symboliky zavedené v odstavci 3.3.4 Reprezentace a řešení úlohy pomocí AND/OR grafu) platí, že jsou ponechány beze změny. Neznámým prvkům ($v(h) = u$) je přiřazena hodnota *t*. Přes takto definované heuristické protějšky určitého vrcholu je v bodě 2) – II spuštěn algoritmus omezené iterace hodnot (limited value iteration), čím se získá heuristická hodnota vrcholu *v*. S touto vypočítanou hodnotou se v algoritmu předepsaným způsobem pracuje. Algoritmus omezené iterace hodnot popisuje například literatura (Shani, Pineau, Kaplow, 2013).

Nejvýznamnější výhoda algoritmu PAO* se nachází v části šíření změn cen (Algoritmus 2 - bod III). Na rozdíl od klasického algoritmu AO* , PAO* nešíří změny cen pouze od předků k rodičovským vrcholům, ale také bokem k sousedům (v celém AND/OR grafu) a zároveň směrem dolů k potomkům. Výsledkem tohoto řešení je plné využití všech

získaných informací, které zlepší následné rozhodování ve všech fázích plánovacího procesu. Díky této klíčové vlastnosti algoritmu PAO* může být pro rodičovský vrchol určeno nejpřesnější ohodnocení a zároveň může být vybrán optimální potomek pro rodičovský vrchol. Algoritmus PAO* má prokázanou o několik řádů vyšší účinnost než klasický algoritmus AO*, přičemž je stále zaručeno nalezení optimálního řešení. (Ferguson, Stentz, 2004b)

3.4 Stochastický problém hledání nejkratší cesty s dynamickým učením

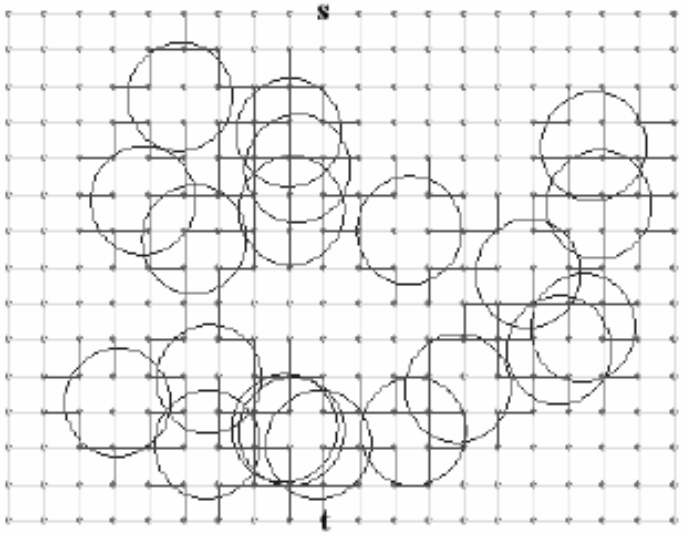
Tato podkapitola se zaměřuje na stochastický problém hledání nejkratší cesty doplněný o schopnost dynamického učení. Tato plánovací úloha předpokládá prostředí, ve kterém se mohou se zadanou pravděpodobností na zadaných souřadnicích vyskytovat překážky. Pro zjištění, zdali je daný neurčitý bod opravdu překážkou, je přiřazena cena. Do této úlohy lze zavést proměnou, která určí maximální počet neurčitostí na naplánované cestě. Hlavním úkolem v této úloze je najít protokol, který rozhodne, které neurčitosti je vhodné odstranit (neboli zahrnout je do plánované cesty) tak, aby byla nalezena nejkratší cesta ze startu do cíle. Aksakalli (2007) tento problém nazývá stochastické hledání nejkratší cesty s dynamickým učením. Jedná se o drobnou modifikaci problému stochastické scény (Stochastic Obstacle Scene) autorů Papadimitrioua a Yannakakise (1991). Stochastický problém hledání nejkratší cesty dynamickým učením nalézá uplatnění především při pravděpodobnostním plánování cesty. Může se jednat například o navigaci robota ve stochastické oblasti, protiopatření proti minovému poli nebo při adaptivním řízení provozu. (Aksakalli, 2007).

Aksakalli (2007) zavádí sjednocující rámec pro stochastické hledání cesty s dynamickým učením pro spojité i diskrétní nastavení daného problému. Tato práce se zaměří na diskrétní variantu zmiňované úlohy, která je řešitelná pomocí metod typu AO*. Bude zde uveden algoritmus BAO* (Aksakalli, 2007), který rozšiřuje výchozí algoritmus AO*, použitím silnějších prořezávajících metod, díky kterým algoritmus BAO* rychleji nalézá optimální řešení. Algoritmus BAO* využívá dvou mezí (horní a dolní) pro určení délky cesty. Aksakalli (2007) jako dolní mez používá nejkratší cestu z aktuálního bodu do cíle. Tuto dolní mez obvykle používá i standardní algoritmus AO* využitý pro plánování cesty v neurčitém prostředí (Bander, White, 2002). Přitom jak uvádí Bander a White (2002) dolní mez může být dána heuristickou funkcí (například Euklidovskou vzdáleností). Jako horní mez zvolil Aksakalli (2007) délku cesty ze startu do cíle, přičemž v této cestě nejsou zahrnuty neurčité body. Nutným předpokladem pro zavedení a následné uplatnění takto definované horní meze je předpoklad, že vždy existuje cesta ze startu do cíle, na které se nevyskytují neurčité body.

3.4.1 Diskrétní stochastický problém hledání nejkratší cesty s dynamickým učením

V uvedeném modelu stochastického problému plánování nejkratší cesty Aksakalli (2007) předpokládá nejisté překážky, které mají tvar kruhu. Tyto překážky vyznačují regiony, které mají neurčitý charakter. Pro tyto regiony je přiřazena pravděpodobnost, zda je daný region skutečně překážkou. Diskrétní podoba stochastického problému hledání nejkratší cesty s dynamickým učením zavádí neorientovaný graf $G = (V, E)$ s danými vrcholy $s, t \in V$ (start a cíl). Předpokládá se, že existuje funkce $l: E \rightarrow R_{\geqslant 0}$, která přiřazuje délku pro každou hranu. Cílem je najít nejkratší cestu z s do t, přičemž všechny hrany nemusejí být průchozí. Zavádí se stochastické hrany $E' \subseteq E$, pro které se dále pomocí funkce $\rho: E' \rightarrow [0,1)$ pro každou hranu $e \in E'$ přiřadí pravděpodobnost $\rho(e)$, že e není průchozí, nezávisle na ostatních hranách. Všechny hrany $E \setminus E'$ jsou průchozí. Plánovací

algoritmus má možnost dozvědět se, zda je e průchozí, za cenu $c(e)$, která je přidána k délce cesty pomocí funkce $c: E' \rightarrow R_{\geq 0}$. Přes neurčité hrany nelze projít, dokud není známo, zda jsou průchozí. Průchodnost neurčitých hran je určena staticky a nemění se v průběhu průchodu grafu. Cílem plánovacího algoritmu je nalézt optimální protokol ve smyslu nejkratší očekávané cesty. Předpokládá se, že existuje limit možných odstranění neurčitostí *K*. Nalezení zmíněného optimálního protokolu v diskrétním stochastickém prostředí se v literatuře (Papadimitriou, Yannakakis, 1991) označuje jako Problém kanadského cestujícího. Předpokládá se, že vždy existuje cesta ze startu do cíle, aniž by bylo nutné projít neurčitou hranou. Avšak tato cesta může být velmi dlouhá. (Aksakalli, 2007)

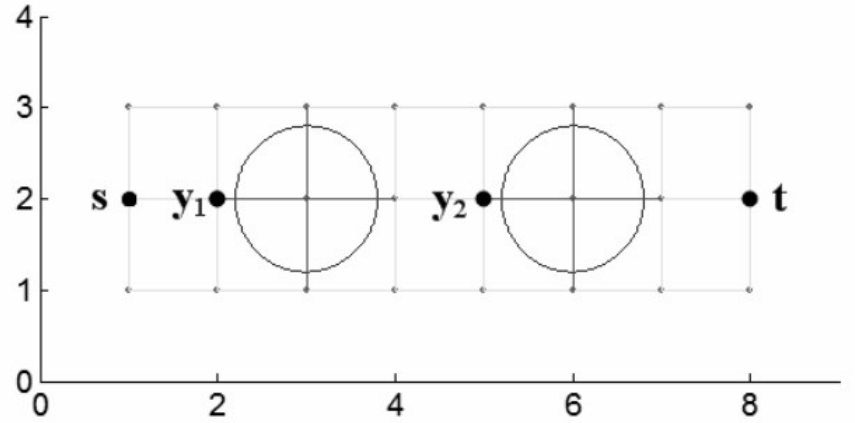


Obr. 7: Příklad scény pro stochastické hledání nejkratší cesty dynamickým učením; uvnitř kružnic se nacházejí neurčité vrcholy (Aksakalli, 2007)

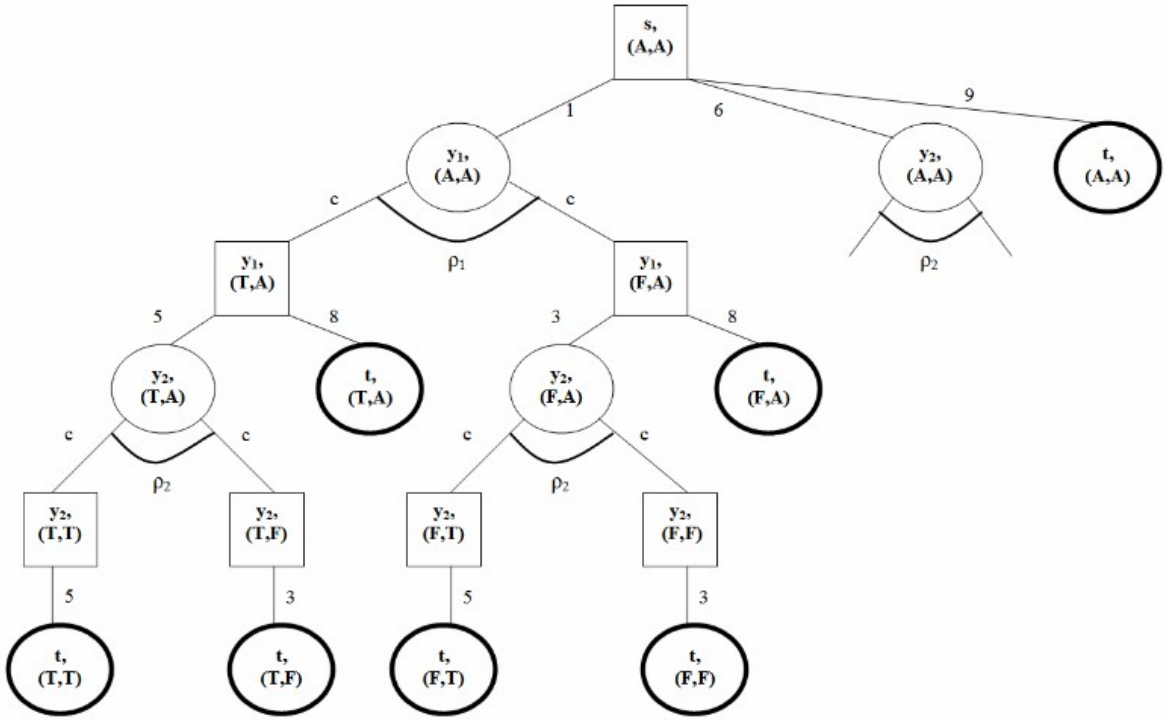
Graf *G* obsahuje vrcholy, které jsou dány dvojicemi celých čísel *i*, *j* tak, že $1 \leq i \leq i_{max}$ a $1 \leq j \leq j_{max}$, kde i_{max} a j_{max} jsou celá čísla. Mezi všemi vrcholy tvaru (i, j) a $(i, j+1)$ existují hrany a taktéž existují pro všechny vrcholy tvaru (i, j) a $(i+1, j)$. Dále v grafu *G* existují dva speciální vrcholy: *s* (start) a *t* (cíl). Cílem plánovacího algoritmu je vytvořit cestu *z* s do *t*, přičemž procházet lze hranami, které neprotínají žádnou jistou i nejistou překážku. Pokud hrana protíná nejistou překážku, může být provedeno zjednoznačnění překážky, a to z kteréhokoliv konce neurčité hrany, jenž je mimo překážku. (Aksakalli, 2007)

Na obrázcích (Obr. 8; Obr. 9) je uveden zjednodušený příklad na diskrétní stochastické plánování s dynamickým učením. V tomto prostředí se nachází start *s*, cíl *t* a dva neurčité regiony. Pro tyto regiony jsou dány pravděpodobnosti, že jsou překážkou ρ_1, ρ_2. Zároveň je neurčitým hranám přiřadí cena zjednoznačnění *c*. Pro jednoduchost se předpokládá, že každý region může být zjednoznačněn pouze ve zvýrazněných bodech (y_1 a y_2). Pro uvedený modelový příklad je na obrázku (Obr. 9) znázorněn odpovídající AND/OR graf. V grafu se nacházejí vrcholy typu OR (čtverce), které představují pozici robota. Vrcholy typu AND (kružnice) odpovídají možným výsledkům akce v rodičovském OR vrcholu. Koncové vrcholy jsou v obrázku (Obr. 9) vyznačeny tučnou kružnicí. V grafu se nachází ještě důležitá doplňující informace: informační vektor *I*. Informační vektor *I*

nese informace o aktuálním stavu možných překážek (stochastických hran). Hodnota informačního vektoru může nabývat hodnot: nejednoznačná (A, ambigous), průchozí (F, false), neprůchozí (T, true) (Aksakalli, 2007b).



Obr. 8: Zjednodušený příklad pro stochastické plánování cesty dynamickým učením (Aksakalli, 2007b)



Obr. 9: AND/OR graf pro příklad na Obr. 8 (Aksakalli, 2007b)

3.4.2 Algoritmus BAO*

Řešení diskrétního stochastického problému hledání nejkratší cesty s dynamickým učením je možné dosáhnout také pomocí klasické metody AO*, která je popsána v podkapitole 3.2 Algoritmus AO*. Aksakalli (2007) však uvádí, že AO* algoritmus dokáže řešit relativně málo příkladů daného problému. V uvedeném modelovém problému je velmi důležité, že hodnota očekávané optimální cesty je vždy omezena horní hranicí označovanou jako nulové riziko (zero-risk). Nulové riziko vyjadřuje délku cesty ze startu do cíle neobsahující neurčité oblastí (v zadaném modelu tato cesta existuje vždy). Algoritmus BAO* do sebe včleňuje horní mez (nulové roziko), díky čemuž významně urychluje vyhledávání optimálního řešení, i přesto, že složitost BAO* není polynomiální. Algoritmus BAO* expanduje pouze vrcholy typu OR. Následníci typu AND a koncové vrcholy typu AND představující přímou cestu do cíle jsou generovány automaticky, ale pouze pokud heuristika těchto vrcholů je menší než hodnota nulového rizika. BAO* nastavuje během expanzního kroku dolní heuristický odhad vrcholů typu OR, které odpovídají překážkám, na hodnotu heuristického ohodnocení jeho rodiče. Tímto krokem je docíleno lepších odhadů. Pokud zbývá pouze jedno možné nejednoznačné místo v současném vrcholu, jenž je určen pro expanzi, pak je vypočítána skutečná hodnota expanzního vrcholu (pomocí vztahů 2 - 3) a odpovídající AND je označen jako koncový. Během zpětného šíření ohodnocení jednotlivých vrcholů algoritmus BAO* kontroluje, zdali nová hodnota daného vrcholu nepřekročila hodnotu nulového rizika. Pokud ano, je daný vrchol vymazán, včetně jeho potomků (Aksakalli, 2007).

1. Stanov hodnotu *zeroRisk*.
2. Vytvoř graf *G,* který se na počátku skládá výhradně z kořenového vrcholu ROOT.
3. Pokud vrchol ROOT není označen jako koncový, pak pokračuj v následujících pod-krocích.
   I. *totalPathDistance* = 0.
   II. *expandNode* = ROOT.
   III. Dokud nebude nalezen neexpandovaný nejlepší neterminální následník, prováděj následující pod-kroky:
      a) Pokud je *expandNode* nelistový AND vrchol, přiřaď jeho potomku, který odpovídá neprůchozímu vrcholu, heuristickou hodnotu jeho rodiče.
      b) *totalPathDistance* += vzdálenost od rodiče *expandNode*
      c) *expandNode* = nejlepší neterminální následovník vrcholu *expandNode*
   IV. Expanduj nejlepší nejnadějnější podstrom. Dočasně vytvoř AND vrchol (cíl, informační vektor *expandNode*). Pokud je jeho hodnota délky cesty ze startu do cíle menší jak *zeroRisk,* přidej dočasný AND vrchol do seznamu následníků *expandNode.*
   V. Pokud ještě zbývají nějaká zjednoznačnění, dočasně vytvoř AND vrcholy odpovídající každé nejednoznačné lokaci náležící dvojznačným stochastickým hranám, které jsou dosažitelné z informačního vektoru patřícímu *expandNode.* A zároveň pro dočasně vytvořené AND vrcholy vygeneruj OR následníky (průchozí i neprůchozí).

Algoritmus 3: BAO - část I. (Aksakalli, 2007b)*

VI. Vypočítej heuristické hodnoty těchto OR vrcholů (zvlášť pro průchozí a neprůchozí). Následně nastav heuristickou hodnotu rodičovského AND vrcholu na $h_{vážené} = \rho \cdot h_{neprůchozí} + (1-\rho) \cdot h_{průchozí}$, kde ρ je pravděpodobnost přiřazená k regionu, ve kterém leží daná stochastická hrana.

VII. Pro každý z AND vrcholů vypočítej heuristickou délku průchodu jako $h_{celkové} = totalPathDistance + h_{cvážené}$. Pokud je $h_{celkové} < zeroRisk$, pak přidej tyto AND vrcholy (včetně jejich potomků) do seznamu následníků *expandNode*. V opačném případě tyto AND vrcholy vymaž. Pokud zbývá pouze poslední zjednoznačnění, vygeneruj následníky pro daný AND vrchol, vypočítej jejich ohodnocení a tento AND vrchol označ jako koncový.

4. Rozšiř nová ohodnocení a aktualizuj graf *G*.
   I. Pokud je *expandNode* vrchol typu OR a nemá žádné potomky, označ ho jako koncový. V opačném případě nalezni potomky vrcholu *expandNode* a najdi vrchol, který má nejmenší heuristické ohodnocení. Aktualizuj u vrcholu *expandNode* heuristickou hodnotu a nejnadějnějšího potomka.
   II. Pokud je *expandNode* vrchol typu AND, nastav heuristickou hodnotu *expandNode* na vážený průměr heuristických hodnot jeho dvou potomků. Pro rodiče vrcholu AND stanov nejnadějnějšího neterminálního potomka. Pokud rodičovský vrchol nemá neterminální potomky, označ ho jako koncový.
   III. Pokud nová heuristická hodnota plus vzdálenost mezi vrcholem *expandNode* a ROOT není menší než *zeroRisk*, pak vymaž *expandNode* včetně jeho potomků.
   IV. Pokud se heuristická hodnota vrcholu *expandNode* nezměnila, pak pokračuj bodem VI. V opačném případě pokračuj bodem V.
   V. Nalezni rodiče vrcholu *expandNode*. Pokud tento rodič neexistuje pokračuj bodem VI. Jinak *expandNode* = rodič vrcholu *expandNode*.
   VI. Pokračuj bodem 3).

Algoritmus 3: BAO- část II. (Aksakalli, 2007b)*

3.5 Stochastické plánování cesty s využitím cyklických AND/OR grafů

V předešlém textu byly analyzovány metody pro plánování cesty v nejistém prostředí s využitím acyklických grafů. Tato podkapitola se zaměří na plánování cesty s využitím cyklických AND/OR grafů. Hansen a Zilberstein (2001) představili upravený algoritmus AO*, který umí najít řešení s cykly. Jedná se o algoritmus LAO* (AO* with loops). Klasický algoritmus AO* nedokáže řešit problémy, které mají cyklické řešení, protože při zpětném šíření cen nahoru do grafu je předpokládáno acyklické řešení. Klíčovým krokem pro zobecnění algoritmu AO* na AO* s cykly je nahradit klasické zpětné šíření cen vybraným algoritmem dynamického programování. Algoritmus LAO* místo klasického zpětného šíření cen využívá algoritmy dynamického programování, jako jsou například iterace hodnot nebo iterace strategií. Popis algoritmů iterace hodnot a iterace strategií lze nalézt například v (Sutton, Barto, 1998).

Jelikož LAO* dovoluje, aby graf řešení obsahoval cykly, dopředné prohledávání z počátečního stavu určené k identifikaci nejlepšího částečného řešení bude ukončeno ve chvíli, kdy bude dosažen stav reprezentující smyčku do již expandovaného stavu. Podobně bude dopředné prohledávání ukončeno pokud bude dosažen cílový, či neterminální listový

stav. Při využití přípustné heuristické funkce nebude žádný stav během procedury LAO* nadhodnocen. Algoritmus LAO* je popsán v textu níže (Algoritmus 4) (Hansen, Zilberstein, 2001).

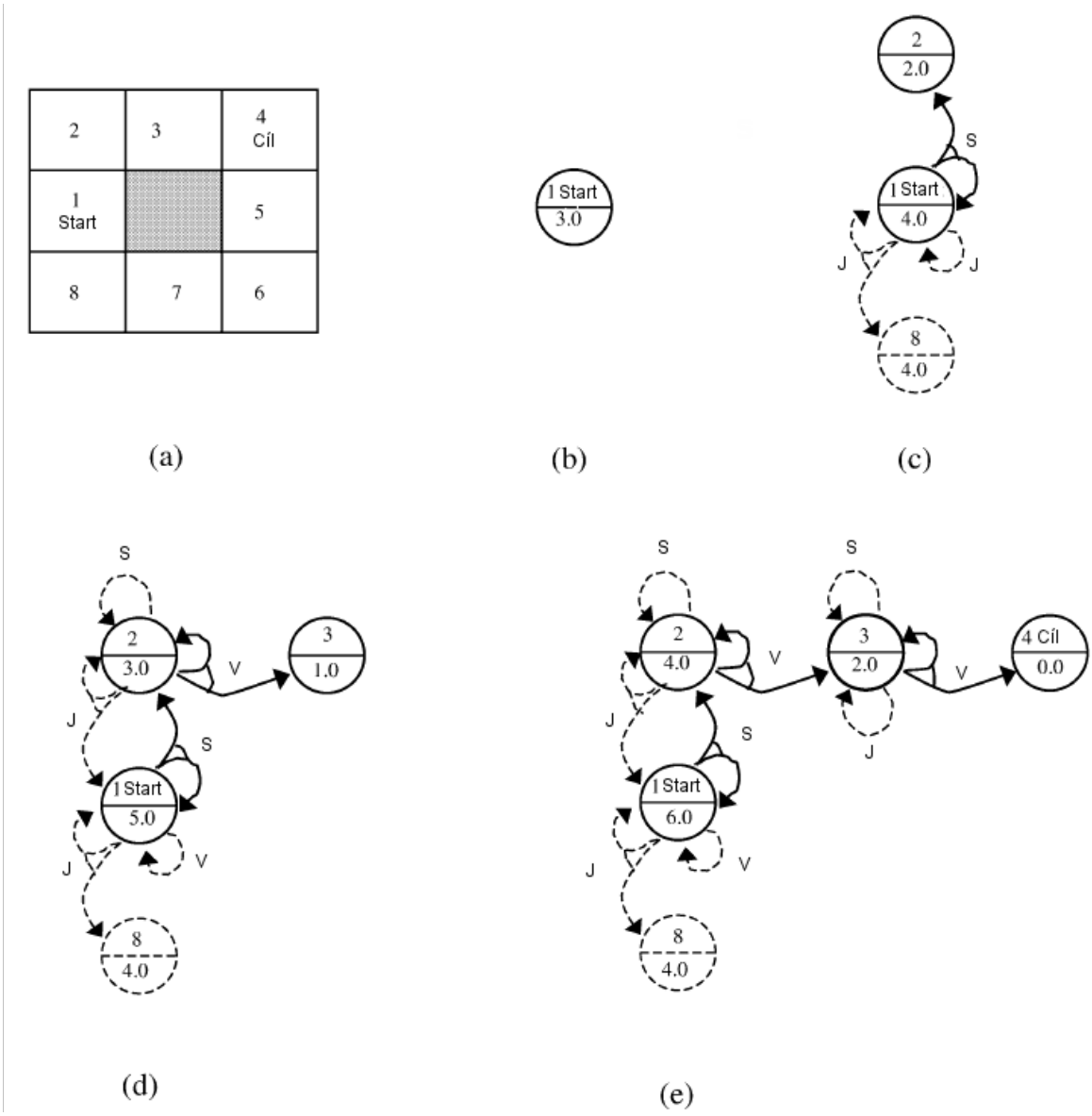
V případě, kdy se v kroku revize cen (Algoritmus 4 / bod 2)-II.-(b)) algoritmu LAO* vypočítá přesná hodnotu stavu pomocí iterace strategií, pak LAO* konverguje, pokud graf nejlepšího řešení nemá žádné neterminální stavy. Tímto je zaručeno, že LAO* konverguje k optimálnímu řešení. Pokud je při revizním kroku využit algoritmus iterace hodnot, pak pokud LAO* nalezne graf řešení, který nemá žádné neterminální stavy, nemusí být nalezené řešení nutně optimální. Situace, kdy je využita jako pod-metoda algoritmus iterace hodnot, je v algoritmu LAO* (Algoritmus 4) ošetřena v bodě 3) (Hansen, Zilberstein, 2001).

1. Součástí explicitního grafu G' je pouze startovní stav s.
2. Dokud graf nejlepšího řešení obsahuje nějaké neterminální neexpandované stavy:
   I. *Expanduj nejlepší částečné řešení*: Expanduj některý neterminální zatím neexpandovaný stav n, který náleží ke grafu nejlepšího řešení a následně přidej nové potomky do G'. Pro každý nový stav i přidaný do G' pomocí expanze uzlu n platí:
      a) Pokud je i cílový stav pak: $f(i) = 0$,
      b) Jinak $f(i) = h(i)$.
   II. *Aktualizuj ceny stavů a označ nejlepší akce*:
      a) Vytvoř seznam S, který bude obsahovat expandovaný stav a všechny jeho předky v explicitním grafu až po označené akční hrany.
      b) Pomocí dynamického programování (iterace strategií nebo iterace hodnot) aktualizuj ceny stavů v seznamu S a následně zvol nejlepší akci pro každý stav.
3. Test konvergence: Pokud byl v kroku 2)-II.-(b) použit algoritmus iterace strategií, pokračuj krokem 4). V opačném případě spusť algoritmus iterace hodnot na stavech v nejlepším grafu řešení. Pokračuj, dokud nebude splněna jedna z sledujících dvou podmínek: a) Pokud je chyba řešení menší než ε, pokračuj bodem 4); b) Pokud se současný nejlepší graf řešení změní tak, že bude obsahovat neexpandovaný stav, pokračuj bodem 2).
4. Vrať graf optimálního řešení.

Algoritmus 4: LAO (Hansen, Zilberstein, 2001)*

Příklad stochastického plánování cesty robota pomocí algoritmu LAO* je zobrazen na obrázku (Obr. 10). Prostředí je zde nahrazeno mřížkou, ve které se nacházejí buňky (stavy – očíslované od 1 do 8) reprezentující polohu robota. Hlavním úkolem je nalézt nejkratší cestu ze startu (stav 1) do cíle (stav 4). V každém stavu jsou dostupné tři možné akce. Jedna akce se pokouší o pohyb na sever (S), druhá akce se pokouší o pohyb na jih (J) a třetí akce se pokouší o pohyb na východ (V). Pravděpodobnost pohybu do požadované buňky je 50% v případě, že daná buňka existuje. Pokud požadovaná buňka neexistuje, poloha robota, tedy jeho stav zůstává nepozměněn. V tomto příkladu je do prostředí tedy zanesena neurčitost akcí. Každá akce v jakémkoliv stavu, kromě cílového, stojí jednu jednotku (cena akce = 1). Přípustná heuristická funkce je dána délkou optimální deterministické cesty ze stavu do cíle. Záložka b) na obrázku (Obr. 10) ilustruje kořenový vrchol, který je na počátku algoritmu vložen do prázdného grafu nejlepšího řešení. Záložka

c) zobrazuje výsledek expanze kořenového stavu. Nejnadějnější akce jsou na záložkách c) až d) označeny nepřerušovanou čárou. Záložka d) zobrazuje výsledek expanze stavu 2, který je nejnadějnějším stavem stavu 1. Výsledek expanze s 3 je znázorněn na záložce d). (Hansen, Zilberstein, 2001)



Obr. 10: Příklad expanze grafu algoritmem LAO pro jednoduché nejisté prostředí znázorněné na položce a). (Hansen, Zilberstein, 2001)*

3.6 Anytime algoritmy a nejisté dynamické prostředí

Algoritmy, které byly doposud v této práci zmíněny, si kladou za cíl nalézt předpokládanou optimální cestu v neurčitém prostředí. Výpočet předpokládané délky optimální cesty může v některých případech trvat dlouho dobu. Například z výsledků práce Aksakalliho (2007) lze zjistit, že pro netriviální úlohu, která obsahuje 20 neurčitých oblastí, se délka předpokládané optimální cesty zjišťuje zhruba osm minut s využitím

algoritmu BAO*. Pokud však robot plánuje svoji cestu v reálném čase, může nastat situace, kdy robot nemá potřebný dostatek času na celý výpočet optimální cesty. V takovém případě je vhodné využít metod anytime algoritmů. Anytime algoritmy, jak uvádí Zilberstein (1996), jsou algoritmy, které postupně vylepšují kvalitu cesty v průběhu času. Pokud nastane situace, kdy robot již nemá potřebný čas na výpočet optimální cesty, pak je v případě metod anytime navráceno nejlepší řešení, které je momentálně k dispozici. Pro plánování cesty v reálném čase je vhodné do plánovacího algoritmu vnést myšlenky anytime metod. V této podkapitole se zaměřím na algoritmy typu AO*, které jsou rozšířeny o myšlenky metod anytime. V první části této podkapitoly bude uveden algoritmus DAO* (Chung, Huang, 2011). Algoritmus DAO* vychází algoritmu AO*, a rozšiřuje ho o myšlenky metod anytime algoritmů. Navíc nabízí možnost rychlého přeplánování trasy, což je při nasazení plánovacího algoritmu do reálných podmínek velmi výhodné. Pře-plánovací algoritmy je vhodné použit pro dynamická prostředí, jak uvádí Berg, Ferguson a Kuffner (2006). V reálném prostředí totiž obvykle nejsou pouze statické překážky. Statické překážky v reálném prostředí mohou být objekty, které obvykle nemění svoji polohu a jsou stále na svém místě (např. dům, zeď apod.). Oproti tomu dynamické překážky svoji polohu mohou měnit (pohybující se člověk, auto, apod.). Využití pře-plánovacího algoritmu se nalezne například v případě, kdy robot při následování naplánované cesty narazí na neočekávanou překážku, která v původním plánu nebyla zahrnuta (například cestu zatarasil sníh). K vytvoření nového plánu cesty lze využít informace z předchozího plánování, které jsou právě schopné využít pře-plánovací algoritmy (Berg, Ferguson, Kuffner, 2006).

3.6.1 Algoritmus WAO*

Heuristické prohledávací algoritmy obvykle spoléhají na heuristickou funkci *f*, která je složena z několika částí (např. $f(n) = g(n) + h(n)$; kde $g(n)$ je cena cesty ze startu do stavu *n*, a $h(n)$ je heuristický odhad skutečné délky cesty $h^*(n)$) (Hansen, Zhou, 2007). Dle (Chakrabarti, Ghose, Desarkar in Chung, Huang, 2011) lze obdobnou dekompozicí heuristické funkce aplikovat i na algoritmus AO*. Vhodnou úpravou heuristické funkce lze dosáhnout účinějšího prohledávání (Chung, Huang, 2011). Chakrabarti, Ghose, Desarkar in Chung, Huang (2011) dokázali, že $h \leq h^*$ je dostačující podmínka pro přípustnost. Pokud je podmínka $h \leq h^*$ uplatněna na všechny stavy a zároveň platí, že h je monotónní, pak algoritmus AO* bude ukončen ve chvíli, kdy nalezne optimální řešení.

Algoritmus WAO* upravuje nákladovou funkci následujícím způsobem dle vztahů (5) (6). V těchto vztazích se vyskytují inflační faktory w a ε. Tyto vztahy jsou ekvivalentní. Hlavní procedura algoritmu WAO* je shodná s výchozím algoritmem AO*. Algoritmus WAO* však využívá již zmíněnou váženou heuristiku při zpětném šíření cen nahoru do grafu. (Chakrabarti, Ghose, Desarkar in Chung, Huang, 2011)

$$f = (1-w)\cdot g + w\cdot h,\ 0 \leq w \leq 1, \tag{5}$$

$$f = g + \varepsilon \cdot h,\ \varepsilon \geq 1 \tag{6}$$

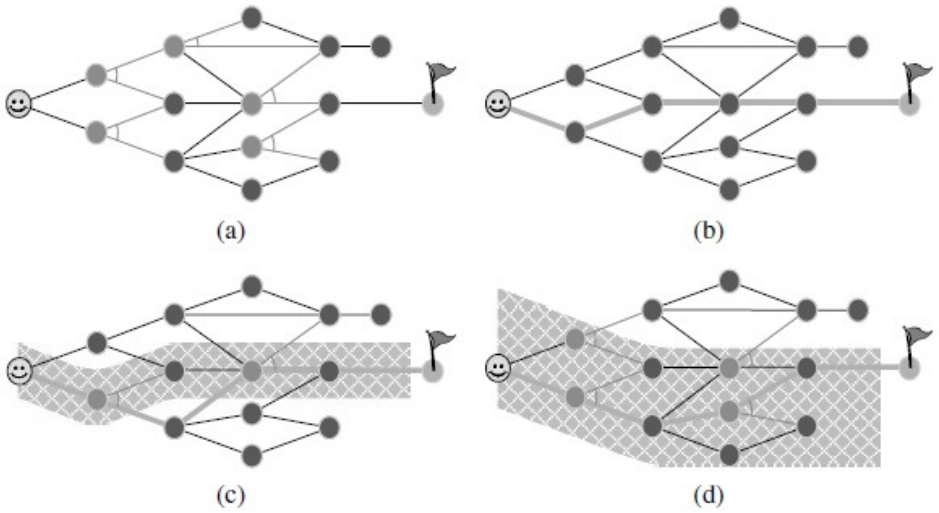
V případě využití vážené heuristiky není zaručeno nalezení optimálního řešení, avšak dochází k výraznému zrychlení vyhledávání. Algoritmus WAO* je základem pro další rozšiřování metod AO*. Z algoritmu WAO* vychází anytime a pře-plánovací algoritmy, například DAO*. (Chung, Huang, 2011)

3.6.2 Algoritmus DAO*

Algoritmus DAO* je určen pro plánování v neurčitém prostředí s možnostmi rychlého přeplánování trasy. Dále tento algoritmus poskytuje vlastnosti, které jsou typické pro metody anytime. V případě, kdy robot tedy nebude mít dostatek času na výpočet optimální cesty, bude robota předán nejlepší plán cesty jaký je v dané době k dispozici. Předání neoptimální cesty v průběhu výpočtu je zajištěna možností ovlivňovat inflační faktor, který se zavádí do heuristiky (viz. 3.6.1 Algoritmus WAO*). Na počátku běhu algoritmu se nastaví inflační faktor ε na vyšší číslo – například na hodnotu 10. Ve chvíli, kdy je nalezeno řešení s tímto inflačním faktorem, algoritmus DAO* sníží o předem stanovený ε a následně pře-plánuje cestu s tímto novým inflačním faktorem. Algoritmus může skončit ve chvíli, kdy nalezne optimální řešení ($\varepsilon = 1$). Případně může také skončit dříve, pokud již nezbývá čas na výpočet optimální předpokládané cesty. V takovém případě algoritmus DAO* navrátí nejnadějnější cestu, jaká je v dané chvíli k dispozici. (Chung, Huang, 2011)

Mapa prostředí obvykle obsahuje mnoho bodů, včetně neurčitostí. Použití algoritmu založeného na metodě AO* nad celou mapou může být velmi časově náročné. Autoři algoritmu DAO*, Chung, Huang (2011), při plánování v neurčitém prostředí nejprve všechny body ve stavovém prostoru označí jako deterministické (Obr. 11b). Následně je nalezena nejkratší cesta ze startu do cíle pomocí deterministického plánovacího algoritmu. Na této cestě jsou zaznamenány všechny neurčité stavy, které se vyskytují v původním stavovém prostoru (Obr. 11a). Následně je hledána nová deterministická nejkratší cesta ze startu do cíle, mimo stochastické body, které byly při předchozím hledání zaznamenány. Po objevení všech stochastických bodů, které mohou ležet na potencionálních nejkratších cestách, může být nalezena očekávaná nejkratší cesta ve stochastickém prostředí pomocí algoritmu DAO*.

Algoritmus DAO* využívá pro hledání strategie výběru stochastických bodů algoritmus prohledávání do šířky. Autoři Chung, Huang (2011) algoritmus DAO* upravují na DDAO*. Algoritmus ddao* pro hledání strategie výběru stochastických bodů využívá algoritmus D* Lite (Koenig, Likhachev, 2005). Podobné předzpracování dat pro metody typu AO*, určené pro plánování v neurčitém prostředí, je použito v práci (Ferguson, Stentz, 2004b). Algoritmus DAO* je uveden níže (Algoritmus 5).



*Obr. 11: Základní myšlenka pro možné předzpracování dat pro metody typu AO**

(Chung, Huang, 2011)

Záloha(*m*)

1. // Aktualizuj potomky, jejichž inflační faktor je nekonzistentní
2. Pro všechny $m_i \in Následovník(m)$,
   I. Pokud platí: $\varepsilon(m_i) \neq \overline{\varepsilon}$
      a) $f(m_i) = g(m_i) + \overline{\varepsilon} \cdot h(m_i)$
3. Pokud m_i je terminální uzel:
   a) Pak označ m_i jako SOLVED a pokračuj bodem 4. V opačném případě proveď b).
   b) Pak označ m_i jako NOT_SOLVED
4. // Aktualizuj uzel *m*
5. Pokud k uzlu *m* náleží množina akcí *A(m)* a odpovídající potomci m_i, nastav hodnotu dle bodu 6. Jinak tento bod přeskoč.
6. $a_{win} = arg\,min_{a \in A(m)}[c_m(a) + \sum_{m_i} p_{mmi}(a) \cdot f(m_i)]$
7. Vyber nejlepší možnou akci a_{win} (upřednostňovávány jsou vrcholy označené SOLVED). Odpovídající konektor označ.
8. $\varepsilon(m) = \overline{\varepsilon}, g(m) = c_m(a_{win}) + \sum_{m_i} p_{mmi}(a_{win}) \cdot g(m_i), h(m) = \sum_{m_i} p_{mmi}(a_{win}) \cdot h(m_i)$
9. $f(m) = g(m) + \varepsilon \cdot h(m)$
10. Uzel *m* označ jako SOLVED v případě, kdy všichni potomci vybrané akce a_{win} jsou označeni jako SOLVED.

Pře-plánování(*m*)

11. Vytvoř seznam *S*. Seznam *S* bude obsahovat uzly, u kterých byly objeveny změny cen hran.
12. Proveď následující pod-kroky pokud seznam *S* není prázdný.
    I. Odstraň ze seznamu *S* uzel *m*, jehož žádný rodič z grafu řešení *G* se nenachází v seznamu *S*.
    II. Proveď proceduru Záloha(*m*)
    III. V případě, kdy byla změněna hodnota $g(m)$ nebo $h(m)$, případně uzel *m* byl označen jako SOLVED, přidej do seznamu *S* rodiče uzlu *m*.

Expanduj(*m*)

13. Přidej nové potomky $m_i \in Následovník(m)$ do grafu *G*.
14. Vypočítej heuristický odhad $h(m_i)$; $g(m_i) = 0$; $\varepsilon(m_i) = \overline{\varepsilon}$; $f(m_i) = g(m_i) + \varepsilon \cdot h(m_i)$
15. Pokud je m_i terminální uzel, označ ho jako SOLVED.

Hlavní procedura(*m*)

16. Zvol $\overline{\varepsilon}$; $\overline{\varepsilon} > 1$
17. Počáteční graf *G* se skládá výhradně z počátečního uzlu *s*. Uzel *s* je označen jako NOT_SOLVED.

Algoritmus 5: DAO - část I. (Chung, Huang (2011)*

18. Pokud uzel *s* není cílový, proveď následující pod-kroky
    I. Pře-plánování(*m*)
    II. Procházej grafem *G* dokud nenarazíš na neexpandovaný nebo terminální uzel *m*.
    III. Pokud je inflační faktor uzlu *m* nekonzistentní, proveď proceduru Záloha(*m*)
    IV. Pokud platí $\varepsilon \neq \overline{\varepsilon}$, pak proveď proceduru Záloha(*m*)
    V. Pokud je *m* listový uzel: Expanduj(*m*) a následně přidej do seznamu U uzel *m*.
    VI. Pokud je *m* označen jako SOLVED: přidej do seznamu U uzel *m.*
    VII. Pokud není seznam U prázdný proveď následující pod-kroky:
        a) Odstraň ze seznamu *U* uzel *m*, jehož žádný rodič z grafu řešení *G* se nenachází v seznamu *U*.
        b) Záloha(*m*)
        c) V případě, kdy byla změněna hodnota $g(m)$ nebo $h(m)$, případně uzel *m* byl označen jako SOLVED, přidej do seznamu *U* rodiče uzlu *m*. Pokračuj bodem VII.
    VIII. Pokud je uzel *s* označen jako SOLVED a zároveň $\overline{\varepsilon} > 1$, pak sniž hodnotu $\overline{\varepsilon}$ a označ *s* jako NOT_SOLVED. Pokračuj bodem 18.
19. Vrať optimální graf řešení *G*.

Algoritmus 5: DAO - část II. (Chung, Huang (2011)*

4 SIMULAČNÍ PROSTŘEDÍ

Hlavním cílem praktické části této práce je implementace a otestování vybraných metod typu AO* určených pro plánování cesty v neurčitém prostředí. Pro otestování plánovacích metod je zapotřebí mít k dispozici simulační prostředí, které testy umožní. Vytvoření simulačního prostředí je jedním z cílů této práce. Mezi základní funkce by měla patří možnost umístění startu a cíle plánované cesty. Další funkcí by měla být možnost přidávat překážky do prostředí, kterými robot nebude moct projít.

Konkrétní simulátor, který je nutné naprogramovat bude určen k otestování implementovaných algoritmů. Pomocí nich bude možno najít cestu ze startu do cíle. Parametry dané cesty (optimálnost, neurčitost na cestě, ...) se budou odvíjet od nastavení příslušného plánovacího algoritmu. Simulátor musí být doplněn o možnost zpracování neurčitosti.

V této práci se zaměřím konkrétně na neurčitost překážek. K neurčitým překážkám bude přiřazena pravděpodobnost, zdali je daná překážka neprůchozí. Konkrétní model plánování v neurčitém prostředí popíši v podkapitole 4.2 Model simulačního prostředí.

V první podkapitole se krátce zaměřím na možnosti vytvoření simulačního prostředí. Dále popíši prostředí, které simulátor vytváří. Ve třetí podkapitole se zabývám možnostmi simulačního prostředí. Čtvrtá podkapitola pojednává o návrhu simulačního programu. Poslední podkapitola obsahuje popis simulačního programu a jeho ovládání.

4.1 Možnosti vytvoření simulačního prostředí

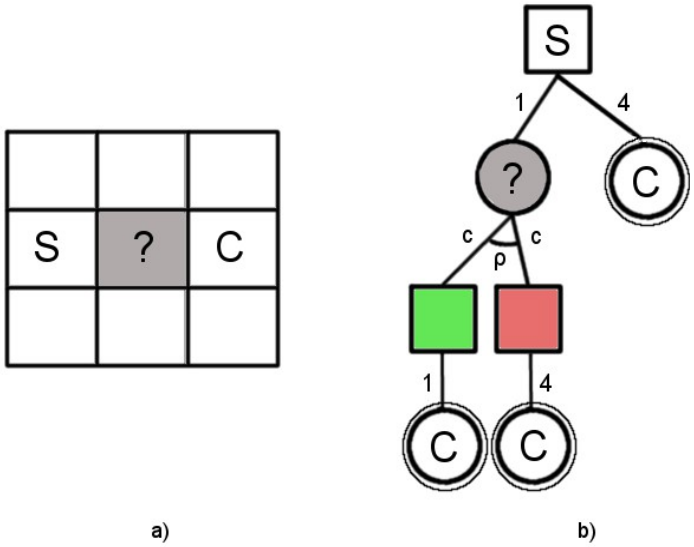
K této problematice lze přistoupit z několika směrů. Jedním přístupem by například bylo vytvoření simulačního prostředí v programovém prostředí MATLAB (The MathWorks, 2015). Vývojové prostředí MATLAB by umožnilo implementaci plánovacích algoritmů, včetně vytvoření grafického uživatelského rozhraní a následné provedení experimentů. Popsaný přístup můžeme najít například v práci Abbadi a Matoušek (2014). Programové prostředí MATLAB má k dispozici mnoho nástrojů (The MathWorks, 2015b), které obvykle obecně urychlují vývoj a testování algoritmů.

Dalším možným přístupem je vytvoření simulačního prostředí s pomocí vysokoúrovňového programovacího jazyka (například C++, C#, Java). Praktickou část této práce jsem se rozhodl vytvořit s využitím jazyka C++. K volbě jazyka C++ mě přivedla zkušenost s tímto jazykem z předchozích projektů. K vývoji celé aplikace, která obsahuje simulační prostředí i plánovací algoritmy, jsem zvolil vývojové prostředí Microsoft Visual Studio Professional 2013. Pro studenty je toto prostředí zdarma dostupné ze serveru Microsoft DreamSpark (Microsoft Corporation, 2015).

4.2 Model simulačního prostředí

Tato podkapitola se zaměří na popis prostředí, které bude vytvořeno plánovacím programem. Plánovacímu programu bude předána mapa, ve které se mohou vyskytovat nejisté překážky. Myšlenka plánování s nejistou mapou vychází z práce (Ferguson, Stentz Thrun, 2004). Předpokládá se, že robot dostane za úkol dostat se ze startovní pozice do pozice cílové. Ke splnění úkolu dostane robot k dispozici mapu, která byla pořízena například bezpilotním letounem. V této mapě se však mohou vyskytovat nejasnosti ohledně průjezdnosti některých cest. Nejasnost ohledně průjezdnosti cesty může být například dána úzkým koridorem, který cesta tvoří. Robot v takovém případě nemusí být schopen úzkým koridorem projet / projít. Existuje ovšem pravděpodobnost, že koridor bude průjezdný / průchodný. V pořízené mapě se také mohou na potenciálně nejkratší cestě

vyskytnout body, které robot nebude schopen jednoznačně prohlásit za překážku. V případě venkovního prostředí by neurčitým bodem mohla být vodní hladina (pokud by robot neuměl plavat). V práci (Ferguson, Stentz Thrun, 2004) je také uváděna neurčitost způsobená nízkým rozlišením mapy. Nízké rozlišení mapy by mohlo být také zdrojem nejasností ohledně průjezdnosti některých cest. Všechny překážky v prostředí budou uvažovány jako statické.



Obr. 12: Jednoduchý příklad modelu plánování v neurčité mapě s využitím AND/OR grafu.

Model plánování v neurčitém prostředí bude vytvořen pomocí AND/OR stromu. Následující definice plánovacího modelu (AND/OR stromu) vychází z práce (Aksakalli, 2007). AND/OR strom T tvořící model neurčitého prostředí je dvojice (V, A), kde V je množina vrcholů stromu a A je množina akcí. Ke stromu T jsou přiřazeny dvě funkce l a p. Funkce $l: A \rightarrow R_{\geq 0}$ přiřazuje vzdálenost do jiného vrcholu (potomka současného vrcholu). Funkce $p: A \rightarrow [0, 1]$ přiřazuje pravděpodobnost ke každé hraně, která vyjadřuje, zda je koncový vrchol dané hrany překážkou. Dále je množina V rozdělena na podmnožinu AND vrcholů (značenou V_A) a podmnožinu OR vrcholů (značenou V_O). Množina V_O bude v modelu vyjadřovat polohu robota. Z vrcholů typu OR povedou hrany do vrcholů AND, případně do listových koncových vrcholů. K hranám vedoucích z OR vrcholu do AND vrcholu (případně do listového koncového) bude přiřazena pravděpodobnost rovna jedné. Vrcholy typu AND budou vyjadřovat neurčitost dané souřadnice (tato myšlenka vychází z práce (Ferguson, Stentz, 2004b)). Vrcholy typu AND budou mít v uvedeném modelu vždy dva potomky typu OR. Jeden tento potomek typu OR bude odpovídat stavu, kdy daná souřadnice je průchozí s pravděpodobností 1 – ρ. Druhý potomek vrcholu AND bude odpovídat stavu, kdy je daná souřadnice neprůchozí s pravděpodobností ρ. Pravděpodobnost ρ stanovující možnost překážky skrytého stavu je svázána s vrcholy typu AND. Každý vrchol AND může mít různou pravděpodobnost ρ. K hranám AND → OR je dále také přiřazena cena c, která vyjadřuje náklady na zjednoznačnění, zda je daný neurčitý bod průchozí, či nikoliv. Jednoduchý příklad založený na popsaném modelu je uveden na obrázku (Obr. 12). Dále je zapotřebí stanovit způsob ohodnocení jednotlivých vrcholů (heuristickou funkci), aby metody typu AO* mohly stavový model prohledat. S

přihlédnutím k práci (Aksakalli, 2007) jsem zvolil pro ohodnocení vrcholů AND/OR stromu heuristickou funkci, která je dána vzdáleností z daného vrcholu do cíle. Přitom je zapotřebí při stanovování heuristického odhadu přihlédnout k rozličným typům jednotlivých vrcholů následujícím způsobem:

- Pokud je stanovován odhad pro kořenový, průchozí OR nebo AND vrchol, budou se dočasně všechny neurčité stavy uvažovat jako *průchozí*. Následně se nad takto nastaveným prostředím spustí deterministický plánovací algoritmus a nalezne se nejkratší cesta ze startu do cíle (na které se všechny skryté stavy uvažují jako *průchozí*). Hodnota délky této cesty je následně přiřazena jako heuristický odhad pro požadovaný vrchol.
- Pokud je stanovován odhad pro neprůchozí vrchol typu OR, budou se dočasně všechny neurčité stavy uvažovat jako *neprůchozí*. Následně se nad takto nastaveným prostředím spustí deterministický plánovací algoritmus a nalezne se nejkratší cesta ze startu do cíle (na které se všechny skryté stavy uvažují jako *neprůchozí*). Hodnota délky této cesty je následně přiřazena jako heuristický odhad pro požadovaný vrchol. Pokud hledaná cesta neexistuje, nastaví se odhad na velké číslo (například polovina hodnoty FUTILITY). Pozn.: hodnota FUTILITY je zavedena v algoritmu AO* - viz. 3.2 Algoritmus AO*.

4.3 Požadavky na simulační program

Tato podkapitola si klade za cíl popsat požadavky na funkcionalitu simulačního programu. Simulační program musí umožnit otestování vybraných metod typu AO*, pomocí kterých bude nalezena očekávávaná optimální cesta v neurčitém prostředí. Základní požadavky na simulační program jsou shrnuty v bodech níže. Body pouze naznačují požadovanou funkcionalitu programu. Každý bod bude dále v textu rozveden a dále specifikován.

a) Simulační program umožní uživateli vytvořit mapu prostředí, ve kterém se bude pohybovat holonomní robot.
b) Simulované prostředí bude reprezentováno mřížkou.
c) Robot se v simulovaném prostředí bude moci pohybovat 8-mi směry. Diagonální pohyb bude umožněn pouze pokud sousední buňky ve směru pohybu budou průchozí.
d) Mapa prostředí bude mít konečný počet buněk, který bude specifikován uživatelem.
e) Do prostředí bude moci uživatel pomocí myši umístit výchozí a cílovou polohu robota.
f) Uživatel bude moci do prostředí umístit deterministické překážky pomocí myši.
g) Do prostředí bude možné také umístit stochastické překážky (skryté stavy). Ke každé překážce bude možné přiřadit číselnou hodnotu pravděpodobnosti, že daný bod je skutečně překážkou.
h) Uživateli bude umožněno překážky (deterministické i stochastické) mazat.
i) Všechny překážky budou statické.
j) Součástí simulačního prostředí bude jednoduchý generátor překážek.
k) Simulační program bude shromažďovat statistiky o daném prostředí. Do statistik se budou také zaznamenávat informace z vyhledávacích metod AO* (jako například: počet expandovaných stavů, délka cesty, čas běhu algoritmu, apod.). Statistiky bude možné exportovat do souboru typu csv.
l) Součástí simulačního programu bude nástroj, který umožní provedení

automatických testů. Testovací nástroj bude využívat generátoru překážek.

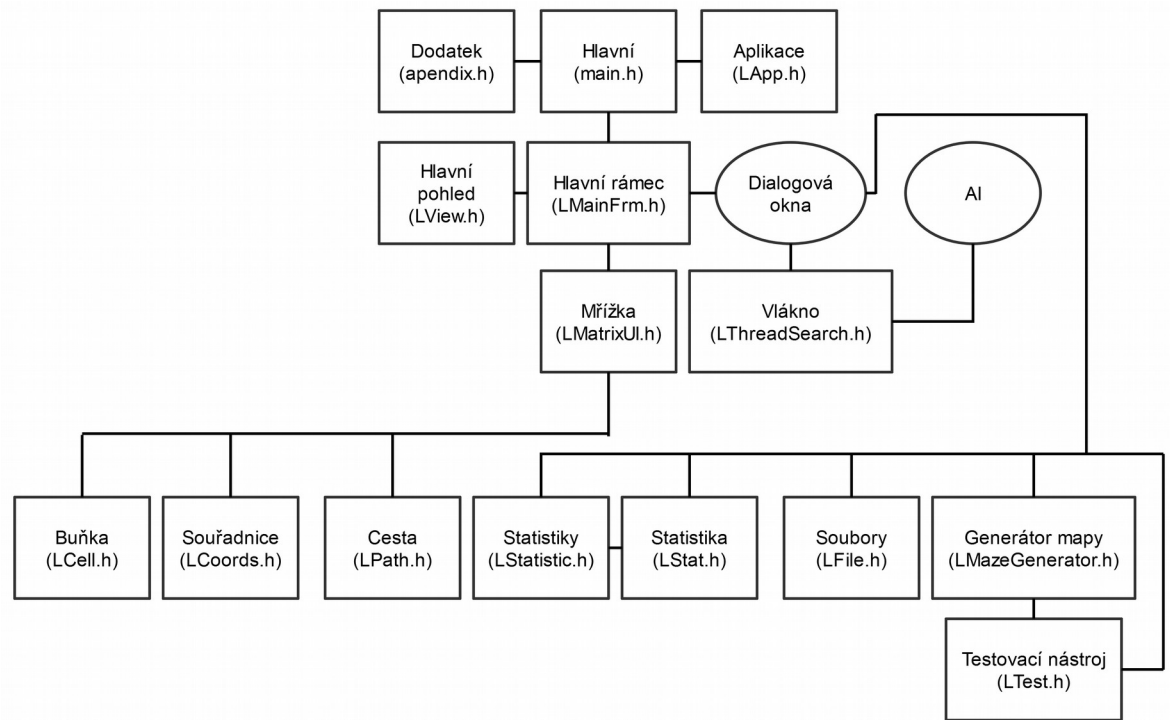
m) Mapu prostředí bude možné uložit do binárního souboru. Uložené prostředí bude možné později znovu načíst do simulačního programu.

4.4 Návrh simulačního programu

V této podkapitole se zaměřím na popis návrhu simulačního programu. Návrh vychází z požadavků, které jsou vytyčeny v podkapitole 4.3 Požadavky na simulační program. Program bude vytvořen pro systém Microsoft Windows 8.1. K programování bude využito vývojové prostředí Visual Studio 2013 a jazyk C++. Pro naprogramování aplikace bude využito rozhraní Windows API (MSDN, 2015) a knihovna Win32++ (verze 7.8) Davida Nashe (Win32++, 2015). Pomocí WinAPI lze vyvinout aplikaci, která bude spustitelná na všech verzích Windows, přitom lze využít funkcí a možností, které jsou jedinečné pro každou verzi. Windows API umožní vytvoření hlavní smyčky programu a zachytávání vstupů od uživatele. Dále se také postará o zobrazení výstupů v okně programu. (MSDN, 2015) Knihovna Win32++ zabaluje WinAPI do objektové podoby (Win32++, 2015).

4.4.1 Zjednodušené schéma programu

Návrh simulačního programu proběhl od shora dolů. Nejprve byly navrženy třídy, které bude program využívat a zároveň byla promyšlena provázanost vybraných tříd. Návrh je proveden tak, aby bylo možné průběžně doplňovat nové funkcionality do programu. Níže uvádím na obrázku (Obr. 13) zjednodušené schéma programu. Jedná se o přehled tříd, které program využívá. Přesnější popis jednotlivých tříd je uveden v podkapitole 4.4.2 Popis tříd. Ve zjednodušeném schématu jsou mimo českých popisků uvedeny také popisky (v závorce), které jsou použity přímo ve zdrojovém kódu u hlavičkových souborů. Uvedené schéma zjednodušuje popis tříd, které se zabývají



Obr. 13: Zjednodušené schéma programu 1 – třídy simulačního programu

prohledáváním stavového prostoru. Položky vztahující se k prohledávání stavového prostoru jsou shrnuty pod filtrem „AI“. Obsah tohoto filtru, včetně popisu je uveden až v kapitole 5 Implementace vybraných plánovacích algoritmů. Podobně je zjednodušen popis použitých dialogových oken (například pro vytvoření nové mapy, apod.). Každé dialogové okno má svoji vlastní třídu. Všechny třídy dialogových oken jsou ve zjednodušeném schématu shrnuty pod filtrem „Dialogová okna“.

4.4.2 Popis tříd

Tato podkapitola se zaměří na stručný popis tříd a hlavičkových souborů programu. Budou zde krátce komentovány třídy a hlavičkové soubory, které utváří uživatelské prostředí simulačního programu. Popis tříd, které se starají o prohledávání stavového prostoru bude přenechán kapitole 5 Implementace vybraných plánovacích algoritmů. Hlavní myšlenky tříd LApp, LMainFrm a LView byly vytvořeny s využitím tutoriálu, který je dostupný s knihovnou Win32++ (Win32++, 2015b).

Dodatek (apendix.h)

Hlavičkový soubor „apendix.h“ obsahuje definice konstant, které se používají v celém projektu.

Hlavní (main.h)

Ve zdrojovém souboru „main.cpp“ je vytvořena instance třídy LApp a následně je spuštěna metoda Run třídy LApp. Metoda Run spustí hlavní smyčku zpráv.

Aplikace (LApp.h)

Třída LApp je potomek třídy CWinApp (knihovny Win32++). Třída CWinApp umožní spouštění hlavní smyčky zpráv, které používá WinAPI. Při vytvoření třídy LApp je zavolána virtuální metoda InitInstance, která je zavolána při vytvoření instance třídy LApp. V přetížené metodě InitInstance (třídy LApp) je zavolána metoda Create třídy LMainFrm, která je zodpovědná za vytvoření hlavního okna aplikace. (Win32++, 2015b)

Hlavní rámec (LMainFrm.h)

Třída LMainFrm je potomek třídy CFrame (z knihovny Win32++). Třída CFrame je zodpovědná za vytvoření okna aplikace včetně menu, toolbaru, statusbaru a hlavního pohledu (view) okna. (Win32++, 2015b)

Třída LMainFrm zachytává zprávy, které uživatel vysílá při použití menu. Na tyto zprávy následně odpovídajícím způsobem reaguje. Například pokud uživatel klikne na položku menu „New“, vybraná metoda třídy LMainFrm vytvoří modální okno, které umožní vytvoření nové mapy.

Mezi další významné metody třídy LMainFrm patří ty, které se starají o posuvníky aplikace. V simulovaném prostředí se může vyskytnout mapa, která je větší než rozměry klientské části okna. V takové situaci se vybrané metody třídy LMainFrm postarají o posouvání mapy. Informace o nastavení okna aplikace (např. rozměry) a simulovaného prostředí (celkový počet buněk, počet zobrazených v okně aplikace, apod.) se předávají třídě LView.

Hlavní pohled (LView.h)

Třída LView je potomkem třídy CWnd (knihovny Win32++). Třída CWnd reprezentuje okno aplikace. CWnd umožňuje vytvoření i destrukci okna a zároveň určuje

jakým způsobem jsou zachytávány zprávy. Třída LView přetěžuje metodu WndProc, která zachytává zprávy okna. Všechny neobsloužené zprávy jsou předávány metodě WndProcDefault. (Win32++, 2015b)

Třída LView umožňuje vykreslení mapy prostředí do okna aplikace (pomocí metody OnDraw). V metodě OnDraw je volána metoda paint třídy LMatrixUI, ve které je vykreslení mapy prostředí naprogramováno. Třída LView dále zachytává akce myši (kliknutí levým tlačítkem myši, držení levého tlačítka myši, pohyb myši) a pomocí daných metod reaguje odpovídajícím způsobem. Například je tímto způsobem vyřešeno umísťování a mazání překážek, počátečního, cílového bodu v mapě.

Dialogová okna

Pomocí dialogových oken jsou vytvořeny následující možnosti aplikace: vytváření nové mapy, upravování nastavení prostředí, statistiky, generování mapy a testovací nástroj. Všechna dialogová okna jsou vytvořena s využitím knihovny Win32++ (Win32++, 2015b) a její třídy Cdialog . Všechna dialogová okna v simulačním programu dědí metody třídy CDialog.

Mřížka (LMatrix.h)

Třída LMatrixUI je třída reprezentující simulované prostředí pomocí mřížky. Mřížka je vytvořena pomocí buněk, které jsou zaštítěny třídou LCell. Všechny buňky mají předem definovaný čtvercový rozměr. Počet buněk v prostředí určuje uživatel při vytváření mapy. Třída LMatrixUI obsahuje metody, které umožňují vykreslování prostředí, vkládání objektů do prostředí (překážek, startu a cíle) a obecně práci s celým simulovaným prostředím. Tato třída je jako jediná v celém projektu vytvořena jako jedináček (MSDN, 2002). Výhodou tohoto řešení je, že celém projektu je možné přistupovat k metodám třídy LMatrixUI. Takto provedená koncepce předpokládá, že program umožní simulovat pouze jedno prostředí v jednu dobu. V případě, že by bylo vyžadováno simulování více prostředí zároveň v jednom programu, byla by uvedená koncepce nevhodná.

Buňka (LCell.h)

Třída LCell reprezentuje jednu buňku prostředí, která je utvářeno pomocí mřížky (třídou LMatrixUI). Každá buňka má následující vlastnosti: souřadnice buňky ve stavovém prostoru, souřadnice pro zobrazení, typ buňky (průchozí, překážka, neznámá), proměnnou pro dočasnou překážku, velikost buňky, pravděpodobnost překážky, barvu buňky, barvu pera (kreslí okraj buňky). Veřejné metody třídy LCell umožňují nastavovat proměnné, které jsou součástí této třídy. Třída LCell má k dispozici veřejnou metodu, která umožňuje vykreslení dané buňky. Této metody pro vykreslení buňky využívá třída LMatrixUI, která vykresluje celé simulační prostředí.

Souřadnice (LCoords.h)

V průběhu celého projektu se využívá v různých situacích souřadnic, které reprezentují buňku v prostředí. Souřadnice mají x-ovou a y-ovou složku. Třída LCoords má metody pro nastavení, získání a porovnání souřadnic.

Vlákno (LThreadSearch.h)

Je vhodné, aby program obsahoval vlákno pro ovládání programu (GUI vlákno) a vlákno pro výpočty (pracovní vlákno) (Win32++, 2015b). Tímto přístupem lze zajistit, aby byl program ovladatelný i během časově náročného výpočtu (například hledání cesty v prostředí, které obsahuje mnoho překážek). Simulační prostředí, které jsem vytvořil

obsahuje GUI vlákno a také pracovní vlákno reprezentované třídou LThreadSearch. Třída LThreadSearch umožňuje spustit vyhledávací algoritmy ve vlastním pracovním vlákně. Při zapnutí vyhledávání je vytvořeno modální okno, které zabrání přístupu a ovládání simulačního prostředí. Vytvořené dialogové okno nabízí možnost ukončení výpočtu a tím zpřístupnění simulátoru. Případně je také toto modální okno ukončeno v případě dokončení výpočtu hledání nejkratší cesty. Třídu LThreadSearch využívá také nástroj pro testování.

Cesta (LPath.h)

Třída LPath umožňuje vykreslení nalezené cesty do prostředí. Vyhledávací algoritmy předají třídě LPath vektor souřadnic nalezené cesty.

Statistiky (LStatistic.h)

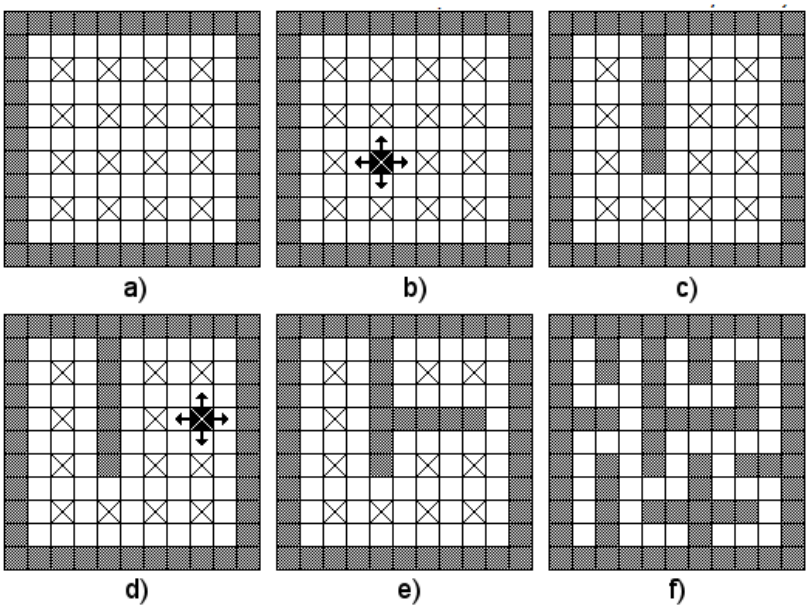
Třída LStatistic slouží k uchovávání statistik, které generují vyhledávací algoritmy (počet expandovaných stavů, délka cesty, doba prohledávání stavového prostoru apod). Součástí jsou také statistiky o simulovaném prostředí (velikost prostředí, počet překážek, počet skrytých stavů). Pro uchování statistik o proběhnutém vyhledávání byla vytvořena dílčí třída LStat.

Soubory (LFile.h)

Třída LFile byla vytvořena, aby měl simulátor možnost ukládat vytvořené prostředí, včetně statistik. Tato třída obsahuje pouze dvě metody, a to pro ukládání do souboru a pro načítání ze souboru. Třída přistupuje k datům aplikace prostřednictvím singletonu LMatrixUI. Získaná data je poté schopna uložit do souboru. Případně je schopna nahrávaná data ze souboru předat třídě LMatrixUI.

Generátor mapy (LMazeGenerator.h)

Pro usnadnění provádění experimentů v simulátoru jsem vytvořil nástroj pro generování překážek. Nástroj pro generování překážek obsahuje třída LMazeGenerator. Algoritmus pro generování překážek jsem vytvořil na základě informací dostupných v (ITnetwork, 2015).



Obr. 14: Ukázka postupu generování bludiště (ITnetwork, 2015).

Algoritmus generování bludiště nejprve vytvoří prázdné prostředí. Okraje prázdného prostředí jsou následně vyplněny překážkami. Dále se na liché souřadnice (s výjimkou okrajů) umístí dočasné značky (Obr. 14a)). V další kroku je vybrána náhodně dočasná značka (Obr. 14b)). Následně se náhodnou volbou zvolí směr, kterým se bude tvořit zeď. Následně se zvoleným směrem začne tvořit zeď, dokud nebude dosaženo překážky (Obr. 14c)). Dále algoritmus generování bludiště opět pokračuje krokem výběru dočasné značky. Algoritmus konči ve chvíli, kdy jsou všechny dočasné značky přeměněny na překážky ((Obr. 14f)). V mapě, která obsahuje překážky vytvořené pomocí uvedeného algoritmu, vždy existuje cesta z libovolných dvou průchozích políček (za následující podmínky). Aby předchozí tvrzení bylo pravdivé, musí mít bludiště lichý rozměr a zároveň souřadnice startu a cíle musí být umístěny také na lichých souřadnicích. (ITnetwork, 2015)

Popsaný algoritmus je v třídě LMazeGenerator rozšířen o další možnosti. Algoritmus umožňuje náhodnou volbou vymazat uživatelem specifikovaný počet dočasných značek. Tímto způsobem lze dle (ITnetwork, 2015) docílit méně komplikované mapy s překážkami. Algoritmus jsem dále rozšířil o možnost generování skrytých stavů. Generování skrytých stavů lze provést dvojím způsobem (záleží na vstupu od uživatele). První způsob generuje náhodně skryté stavy pouze do předem zadaných deterministických překážek. Druhý způsob generuje náhodně skryté stavy libovolně do prostředí (do dosud průchozích buněk i buněk obsahujících překážky).

Testovací nástroj (LTest.h)

Třída LTest zpřístupňuje simulačnímu programu možnosti provádět automatické testy. Testovací nástroj využívá generátoru překážek, pro vytváření prostředí. Nad prostředím následně spouští implementované metody pro hledání optimální cesty. Testování je spuštěné ve vlastním pracovním vlákně (s využitím třídy LThreadSearch). Testování je díky tomu možné kdykoliv ukončit.

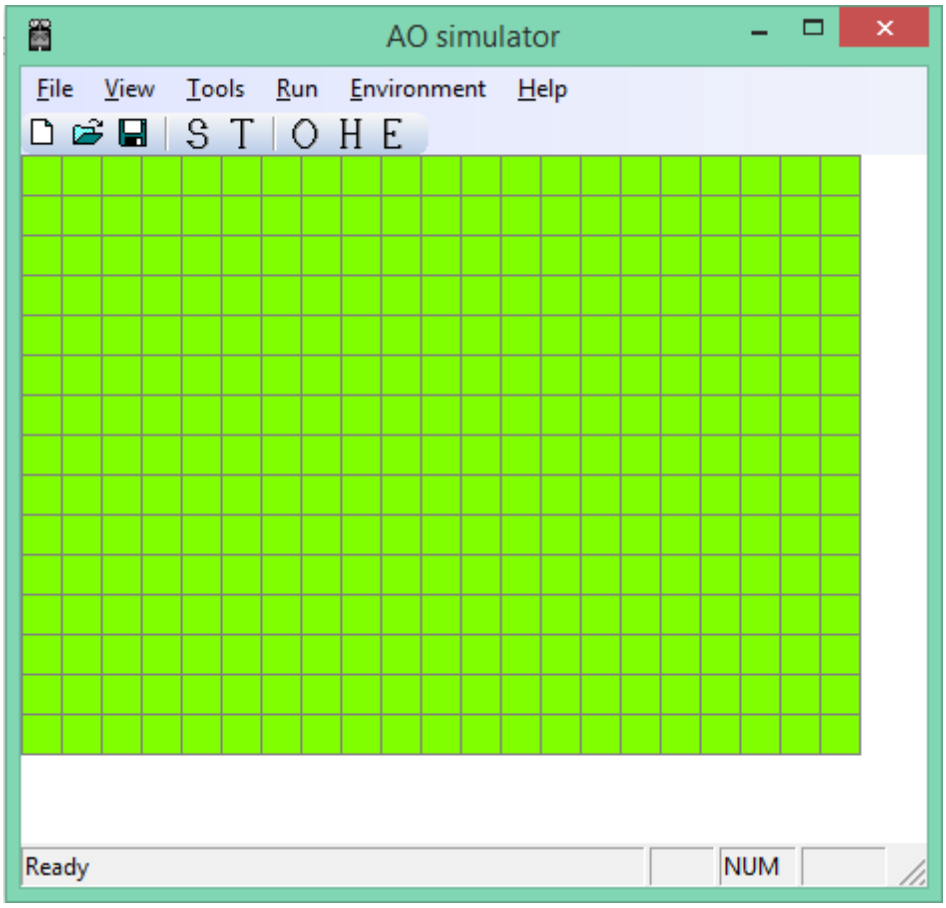
4.5 Popis simulačního programu

V této podkapitole se zaměřím na popis vytvořeného simulačního programu z uživatelského hlediska. Program byl zkompilován pro systém MS Windows (zvlášť pro instrukční sady x86 a x64). Simulační program byl testován na systému MS Windows 8.1.

Simulační program slouží k otestování implementovaných metod (A*, AO*, WAO*). Program umožňuje nalezení optimální cesty v deterministickém prostředí, pomocí algoritmu A*. Hlavním cílem této práce je však plánování cesty v neurčitém prostředí. Simulátor umožňuje naplánování cesty v neurčitém prostředí pomocí algoritmu AO* a WAO*. Do prostředí je možné umístit kromě překážek skryté body. Ke skrytým bodům lze přiřadit pravděpodobnost, že daný bod je neprůchozí. Konkrétní model simulovaného prostředí byl popsán v podkapitole 4.2 Model simulačního prostředí. Implementovaný algoritmus AO* je schopný nalézt očekávanou optimální cestu v neurčitém prostředí. Algoritmus WAO* optimální cestu nalézt nemusí (jedná se o algoritmus AO* s váženou heuristikou). Avšak algoritmus WAO* umožňuje nalézt cestu rychleji. Teoretický rozbor zmíněných metod typu AO* byl proveden v kapitole 3 Plánování cesty v neurčitém prostředí.

Program umožňuje vytváření prostředí v 2D mřížce. Dostupný je také generátor překážek. Vytvořené prostředí je možné uložit do souboru pro pozdější opětovné nahrání. Součástí programu je testovací nástroj, který slouží k provedení automatických testů, které specifikuje uživatel. Program shromažďuje statistiky z prohledávacích metod. Statistiky je možné ukládat do souboru typu „csv“. Prostředí programu (popisky menu, hlášení, …) je popsáno pomocí anglického jazyka. Prostředí simulačního programu „AO sim“ je

zobrazeno na obrázku (Obr. 15: Aplikace „AO sim“ po spuštění.).



Obr. 15: Aplikace „AO sim“ po spuštění.

4.5.1 Nové prázdné prostředí

Vytvoření nového prázdného prostředí pro simulaci je možné prostřednicím menu aplikace: File → New. Případně je možné využít první položky zleva na toolbaru (obrázek prázdného listu – viz Obr. 15: Aplikace „AO sim“ po spuštění.). Všechny položky v nově vytvořeném prostředí (buňky zelené barvy) jsou průchozí (nejsou překážkami) a zároveň se nejedná o startovní, či cílové pozice.

4.5.2 Otevírání a ukládání prostředí

Aplikace umožňuje uložení vytvořeného prostředí. Aplikace uloží do souboru přesnou kopii prostředí (všechny překážky, skryté stavy, start a cíl), všechny statistiky i nalezené cesty.

Pro uložení prostředí do souboru může uživatel využít položku menu File → Save. Pro uložení do souboru je v programu také určena třetí položka toolbaru zleva (obrázek diskety – viz Obr. 15: Aplikace „AO sim“ po spuštění.).

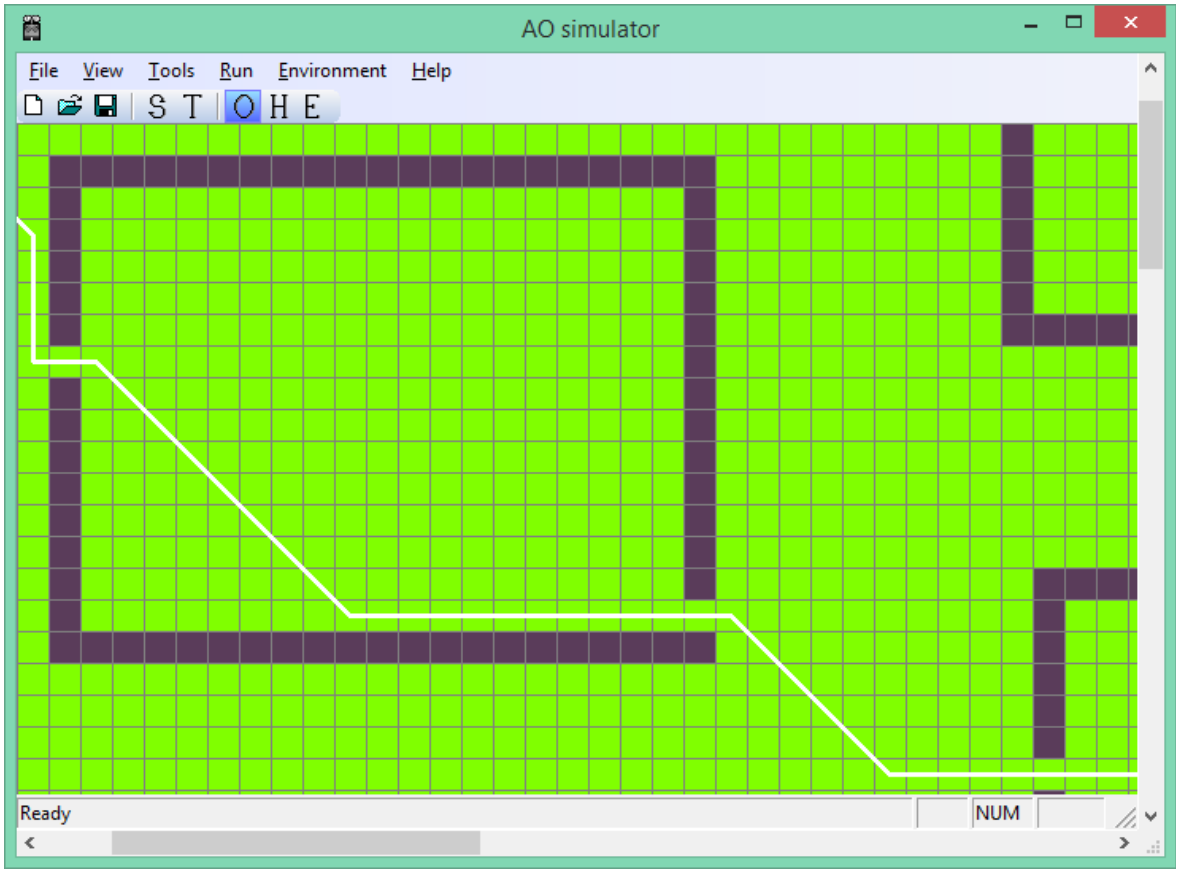
V případě, kdy uživatel vyžaduje otevření již dříve uloženého souboru lze postupovat následujícím způsobem: volba menu: File → Open; případně lze pro otevření souboru použít druhou položku toolbaru zleva (obrázek složky – viz Obr. 15: Aplikace

„AO sim“ po spuštění.)

Aplikace ukládá data do binárního souboru typu *.gld. Jedná se o typ souboru vytvořený speciálně pro účely této aplikace. Soubory jiného typu nelze v simulačním prostředí „AO sim“ otevřít.

4.5.3 Posuvníky aplikace

Vytvořené prostředí pomocí aplikace AO sim může mít rozměr větší než jsou rozměry klientské části popisované aplikace. V takové případě aplikace zobrazí horizontální a vertikální posuvníky, které umožní uživateli posouvat a prohlížet celé prostředí.



Obr. 16: Posuvníky prostředí aplikace AO sim.

4.5.4 Ruční vytváření prostředí

Pro utváření simulačního prostředí jsou v programu k dispozici nástroje na toolbaru. Konkrétně se jedná o položky toolbaru s názvy „S“, „T“, „O“, „H“, „E“ (viz Obr. 15: Aplikace „AO sim“ po spuštění.).

Pro *vložení překážky* do prostředí je zapotřebí na toolbaru zvolit položku „O“. Následně je možné jednotlivé buňky prostředí přetvářet na překážky pomocí myši. Umístění překážky lze provést buď jedním kliknutím myši na vybranou buňku nebo lze překážky kreslit přidržením levého tlačítka a následným tahem myši. V případě, že uživatel již nechce kreslit překážky, pak zruší volbu „O“ na toolbaru. Překážky mají v prostředí tmavě fialovou barvu.

Pro *vložení startu* je zapotřebí na toolbaru zvolit položku „S“. Následně lze kdekoliv do prostředí vložit pomocí levého tlačítka myši vložit start. Do prostředí lze vložit pouze jedenu startovní pozici robota. V případě pokusu o vícenásobné vkládání startu je vždy předchozí start vymazán a nahrazen průchozí buňkou. Buňka start má v prostředí modrou barvu.

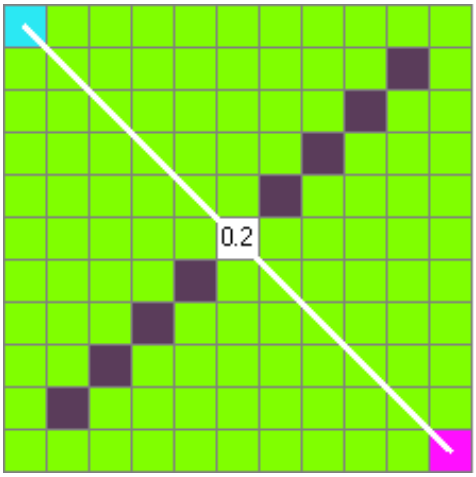
Pro *vložení cílové pozice* robota je zapotřebí na toolbaru zvolit položku „T“. Následně platí analogicky informace, které jsou uvedené předchozím odstavci (*vložení startu*). Buňka cíl má v prostředí růžovou barvu.

Pro vložení *skrytého stavu* (neurčité buňky) je zapotřebí na toolbaru zvolit položku „H“. Následně je možné umístit skrytý stav do libovolné buňky pomocí levého tlačítka myši. Výchozí pravděpodobnost, že daný skrytý stav je překážkou, je nastavena na hodnotu 0.5 (50%).

Hodnotu pravděpodobnosti skrytého stavu je možné editovat následujícím způsobem: přidržením levé klávesy CONTROL a následným kliknutím levého tlačítka myši na skrytý stav. Uživateli je následně umožněno pomocí dialogového okna editovat hodnotu pravděpodobnosti překážky skrytého stavu. V případě, že uživatel chce provést editaci hodnoty pravděpodobnosti skrytého stavu při vkládání, je možné postupovat následujícím způsobem: Předpokládá se vybrání možnosti „H“ na toolbaru. Následně může uživatel současným přidržením levé klávesy CONTROL a stisknutím levého tlačítka myši nad vybranou buňkou, vložit do vybrané buňky skrytý stav. Přitom v tomto případě bude uživateli rovnou nabídnuta možnost editace hodnoty pravděpodobnosti skrytého stavu.

Pro *vymazání buňky* neboli její přeměnu na typ průchozí lze využít nástroj toolbaru označený jako „E“. Po vybrání nástroje erase („E“) na toolbaru je možné pomocí levého tlačítka myši mazat buňky prostředí (přeměňovat buňky na průchozí).

Příklad prostředí, které lze vytvořit popsaným způsobem je zobrazen na obrázku (Obr. 17: Příklad ručně vytvořeného prostředí v programu AO sim.)



Obr. 17: Příklad ručně vytvořeného prostředí v programu AO sim.

4.5.5 Automatické vytváření prostředí

Aplikace AO sim obsahuje nástroj pro automatické generování prostředí. Tento nástroj je dostupný v menu: nabídka Tools → Maze Generator. Okno, které se nachází pod

uvedenou volbou je zobrazeno na obrázku (Obr. 18: Generátor překážek dostupný v aplikaci AO sim.).

V generátoru je možné nastavit velikost prostředí (položka *Number of X/Y cells*) a počet skrytých stavů (*Number of unknown cells*). Aby generátor překážek fungoval řádně, je zapotřebí zadat lichý rozměr prostředí a zároveň start a cíl umístit také na liché souřadnice (vysvětleno v podkapitole: 4.4.2 Popis tříd, odstavec „Generátor mapy“).

Volba popsaná v generátoru jako *Numbers of base → obstacle transformation* (dále zkratka OT) umožňuje zjednodušit prostředí (z hlediska hledání cesty). Pokud je pole OT rovno nule, bude vygenerovaná mapa obsahovat mezi dvěma průchozími buňkami vždy pouze jednu cestu. Pokud hodnota OT bude větší jak nula, pak předchozí věta nemusí být pravdivá. Čím vyšší bude hodnota OT, tím méně překážek bude prostředí obsahovat.

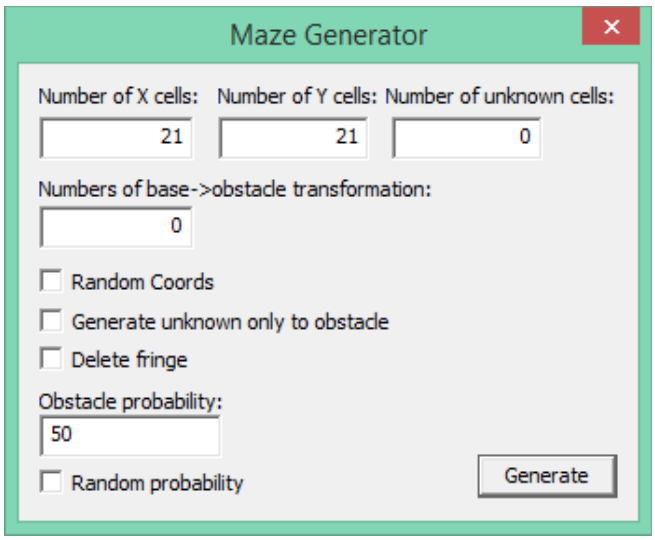
Volba *Random Coords* v generátoru zajistí náhodné vygenerování souřadnic startu a cíle v prostředí. Pokud tato volba není zvolena, musí uživatel umístit souřadnice startu a cíle do prostředí ručně.

Volba *Generate unknown only to obstacle* zajistí vygenerování skrytých stavů pouze do překážek. Pokud je tato volba zapnuta, pak ve vygenerované mapě bude vždy existovat cesta ze startu do cíle, aniž by se muselo projít skrytým stavem (pokud je splněna podmínka lichého rozměru mapy).

Vygenerovaná mapa v základním nastavení obsahuje vždy po svých okrajích překážky. Tento okraj je možné vymazat pomocí volby *Delete fringe*.

Generátor lze nastavit tak, aby všechny skryté stavy měly shodnou pravděpodobnost, že jsou neprůchozí. To lze provést pomocí volby nastavení konstantní hodnoty v položce *Obstacle probability*. V případě, že uživatel vyžaduje náhodné rozdělení pravděpodobnosti pro jednotlivé skryté stavy, uživatel zaškrtne volbu *Random probability*.

Vygenerování mapy umožňuje tlačítko *Generate*. Pokud uživatel mapu vygenerovat nechce, lze dialogové okno generátoru zavřít křížkem. Při použití generátoru je zapotřebí dát pozor, zda je uložena předchozí mapa, jinak bude bez potvrzení smazána.



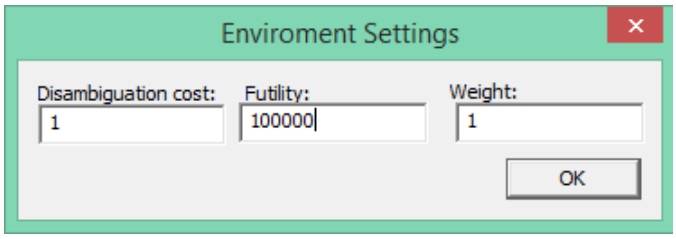
Obr. 18: Generátor překážek dostupný v aplikaci AO sim.

4.5.6 Nastavení prostředí

Vlastnosti prostředí lze nastavit pomocí položky menu Environment → Settings. V dialogovém okně nastavení prostředí se nachází tři položky.

Položka *Disambiguation cost* nastavuje cenu zjednoznačnění skrytého stavu. Více informací o ceně zjednoznačnění lze nalézt například v 4.2 Model simulačního prostředí. Položka *Disambiguation cost* hraje významnou roli při hledání cesty (prohledávání AND/OR grafu) pomocí metod AO*. Položka *Futility* nastavuje maximální možnou hodnotu hledaného řešení. Více informací o této položce lze nalézt v podkapitole 3.2 Algoritmus AO*. Poslední položkou dostupnou v nastavení prostředí je políčko *Weight*. Hodnota políčka *Weight* nastavuje váhu pro algoritmus WAO*, který je v prostředí implementován. Význam váhy pro algoritmus WAO* je popsán v 3.6.1 Algoritmus WAO*.

Upozornění pro uživatele: tato nastavení nejsou ukládána ani načítána ze souborů typu *.gld (obsahující uložené prostředí). Pokud uživatel vyžaduje ověření proběhlých testů z uloženého souboru, pak je nutné po načtení příslušného souboru znovu provést nastavení prostředí.



Obr. 19: Nastavení prostředí aplikace AO sim.

4.5.7 Plánování cesty

Tato podkapitola se zaměří na popis ovládání simulačního programu z hlediska plánování cesty. Tato podkapitola bude předpokládat, že uživatel vytvořil alespoň prázdné prostředí, případně toto prostředí doplnil o překážky a skryté stavy. Aby plánování cesty mohlo proběhnout, musí se v prostředí nacházet výchozí a cílová pozice robota. V případě, že jsou popsané předpoklady splněny, pak může uživatel volbou v menu „Run“ vybrat algoritmus, který naplánuje cestu ze startu do cíle. Pro plánování jsou v menu „Run“ k dispozici následující prohledávací algoritmy: A*, AO* a WAO*. Algoritmus A* je schopen nalézt optimální cestu ze startu do cíle při neuvažování skrytých stavů. Algoritmy AO* a WAO* jsou naopak určeny k plánování cesty v neurčitém prostředí.

Při vyhledávání cesty (tedy po kliknutí na vybranou položku menu „Run“) se objeví dialogové okno, které uživateli hlásí aktuální stav hledání. Spuštěný algoritmus, který je určen pro plánování cesty lze přerušit ve zmíněném dialogovém okně pomocí tlačítka „Stop“.

Uživatel bude po ukončení plánovacího algoritmu informován o úspěchu, či neúspěchu plánování cesty. V případě, že cesta byla nalezena, je možné naplánovanou cestu zobrazit pomocí v prostředí pomocí volby menu View → Path → A*, AO*, WAO*.

4.5.8 Statistiky

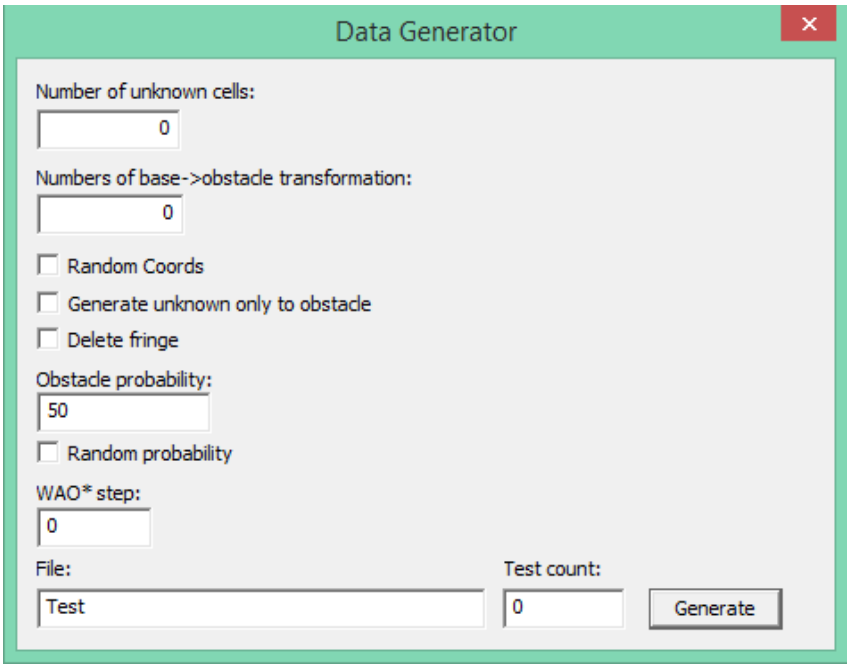
Aplikace AO sim shromažďuje statistiky o aktuálním prostředí a proběhnutých

plánováních. Statistiky jsou dostupné pod volbou menu Tools → Statistics. Statistiky je možné exportovat do externího souboru pomocí tlačítka export, které se nachází v dialogovém okně statistik. Vytvořený soubor je typu csv. Pokud uživatel vytvoří nový soubor, který obsahuje statistiky, bude v tomto souboru v prvním řádku vytvořena hlavička a v druhém řádku budou shromážděny odpovídající statistiky. Uživatel má však možnost vybrat i již vytvořený csv soubor (musí se jednat o soubor csv, který byl vytvořen programem AO sim). V tomto případě nebude vytvářena nová hlavička, ale budou pouze přidány nové statistiky na poslední řádek csv souboru.

4.5.9 Testovací nástroj

Aplikace AO sim disponuje nástrojem, který dovede provést automatické testy. Tento nástroj je dostupný pod volbou menu Tools → Test. Testovací nástroj obsahuje možnosti, kterými disponuje generátor prostředí (4.5.5 Automatické vytváření prostředí). Testovací nástroj však dále obsahuje pole pro určení názvu souborů, které budou programem generovány (položka *File*). Do položky *File* je možné napsat název souboru. V takovém případě se soubory budou generovat ve složce, kde se nachází aplikace AO sim. Pokud uživatel chce specifikovat cestu ke generovaným souborům, pak je před název souborů nutné zadat cestu k dané složce (např. „C:/Dokumenty/NazevGenerovanychSouboru“. Další volbou je stanovení počtu testů (položka *Test count*).

Možné je také nastavit krok pro inkrementaci váhy algoritmu WAO* (položka *WAO* step*). Krok inkrementace váhy WAO* může být desetinné číslo. Pokud je položka *WAO* step* nastavena na hodnotu nula, proběhne algoritmus WAO* s hodnotou váhy, která byla uživatelem zvolena v nastavení prostředí (Menu → Environment → Settings). Pokud je však krok váhy algoritmu větší jak nula, bude se algoritmus WAO* nad daným prostředím opakovat, dokud bude nalezená cesta optimální. Přitom při každém novém opakování bude váha algoritmu WAO* navýšena o krok daný položkou *WAO* step*. Využití této možnosti testovacího nástroje bude demonstrováno na příkladě v kapitole 6 Ověřovací experimenty.



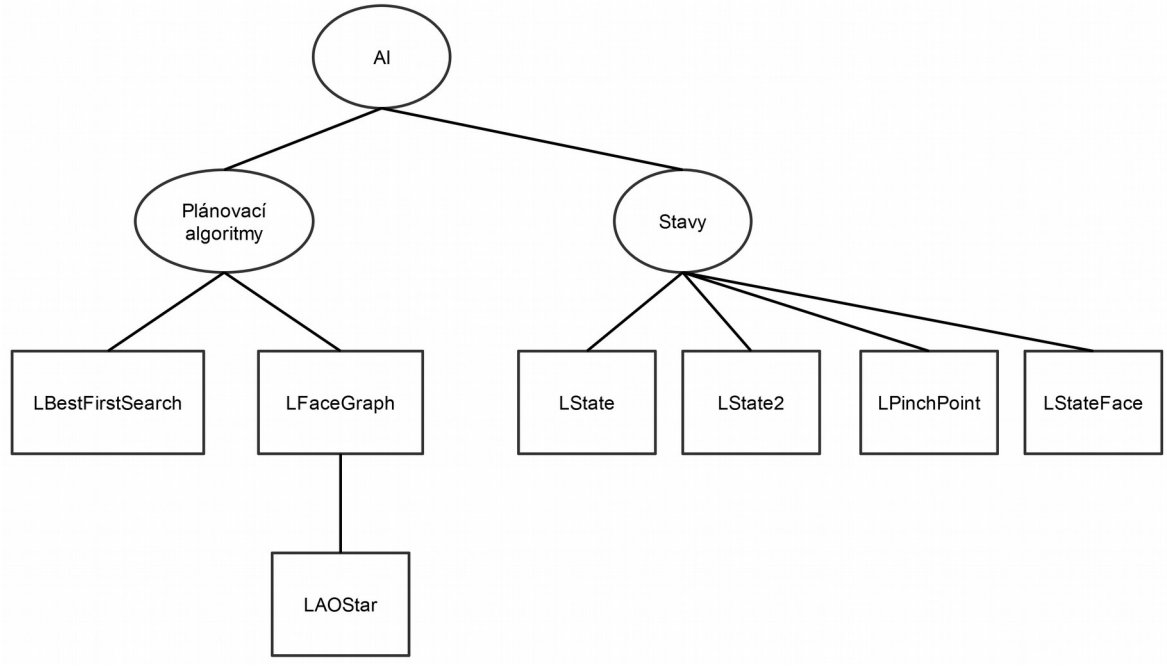
Obr. 20: Testovací nástroj aplikace AO sim.

5 IMPLEMENTACE VYBRANÝCH PLÁNOVACÍCH ALGORITMŮ

V této kapitole se zaměřím na popis implementace plánovacích algoritmů, které využívá program AO sim. Model prostředí, které program simuluje je popsán v podkapitole 4.2 Model simulačního prostředí. V programu jsou implementovány algoritmy pro hledání optimální cesty v deterministickém i stochastickém prostředí. Pro plánování cesty v určitém prostředí jsem zvolil algoritmus A*. Pro nalezení předpokládané optimální cesty v neurčitém prostředí jsem použil algoritmy AO* a WAO*. Implementaci algoritmů jsem provedl ve vývojovém prostředí MS Visual Studio Professional 2013. Ve stejném vývojovém prostředí je napsán celý simulační program.

Návrh tříd náležejících k plánovacím algoritmů navazuje na schéma uvedené na obrázku (Obr. 13: Zjednodušené schéma programu 1 – třídy simulačního programu) v podkapitole 4.4.1 Zjednodušené schéma programu. Rozšíření zmíněného schématu o návrh tříd určených pro plánování je uvedeno v následující podkapitole 5.1 Návrh plánování.

5.1 Návrh plánování



Obr. 21: Zjednodušené schéma programu 2 - třídy pro plánovací algoritmy.

Třídy vztahující se k plánovacím algoritmům lze rozdělit do dvou hlavních kategorií – třídy hlavní a podpůrné. K hlavním třídám patří LBestFirstSearch a LAOStar. Hlavní třídy obsahují algoritmy určené prohledávání stavového prostoru. Třída LBestFirstSearch obsahuje algoritmus A*, který je v aplikaci využíván pro deterministické hledání nejkratší cesty. Třída LAOStar obsahuje algoritmus AO*, který je v aplikaci použit pro plánování cesty ve stochastickém prostředí. Hlavní třídy čerpají z funkcionalit

tříd podpůrných. Jedná se zejména o třídy utvářející stavový prostor (stavy). Stručný popis jednotlivých tříd je proveden v následující podkapitole 5.2 Popis tříd.

5.2 Popis tříd

5.2.1 LBestFirstSearch

Třída obsahuje implementaci algoritmu A*. Třídu lze využít přímo pro naplánování cesty ve vytvořeném simulátoru. Metody tříd LBestFirstSearch slouží k vyhledání deterministické cesty v simulovaném prostředí. Hlavním účel LBestFirstSearch se však nalézá u tříd zabývajících se stochastickým plánováním cesty pomocí metod AO*. Algoritmus A* slouží jako pod-metoda pro algoritmy typu AO*. Algoritmus A* jsem do třídy LBestFirstSearch implementoval s využitím literatury (Březina, Dvořák, 2013b). V následujícím textu jsou stručně popsány veřejné metody třídy LBestFirstSearch.

Metoda search

Pro nalezení optimální cesty v mřížce, kterou tvoří třída LMatrixUI, slouží metoda search. Třída LMatrixUI je v projektu vytvořena jako jedináček (MSDN, 2002). Současný návrh aplikace předpokládá, že se v simulátoru vyskytuje současně nejvýše jedno simulované prostředí.

Metoda search je v rozhraní zdrojového kódu zapsána v následujícím tvaru:

```
int search(LCoords sSource, LCoords sTarget, bool goPinchPoints, HANDLE &stopRequest);
```

Metoda search vyžaduje následující vstupy:

- souřadnice startu a cíle,
- proměnnou typu bool goPinchPoints, která určuje zda má algoritmus A* uvažovat skryté stavy dočasně jako průchozí,
- handle umožňující ukončení algoritmu uživatelem (pracovního vlákna, ve kterém je algoritmus spuštěn).

Metoda search vrací hodnotu typu int, která v daném případě vyjadřuje délku nalezené cesty. V případě, že cesta nebyla nalezena, je návratová hodnota rovna nule.

Metoda backtrack

Metoda backtrack slouží k nalezení strategie plánování cesty. Metodu backtrack je vhodné zavolat až po ukončení metody search. Metoda backtrack nalezne v seznamu CLOSED (který používá algoritmus A*) cílový stav. Cílový stav obsahuje index předka (stavu), který předchází cílovému stavu na optimální cestě. Každý nalezený stav pomocí backtrackingu obsahuje index předka, který daný stav předchází na optimální cestě.

Metoda backtrack je v rozhraní zdrojového kódu zapsána v následujícím tvaru:

```
vector<LCoords> backtrack(HANDLE &stopRequest);
```

Metoda backtrack vyžaduje následující vstup:

- adresu handlu, který umožňuje uživateli backtracking přerušit (pracovního vlákna, ve kterém je algoritmus spuštěn).

Metoda backtrack vrací vektor souřadnic, které dohromady utvářejí plán optimální cesty. V případě, že cesta neexistuje, pak uvedená metoda vrátí prázdný vektor souřadnic.

Metoda getPinchPointsCoords

Metoda getPinchPointsCoords je určena k nalezení skrytých stavů na optimální cestě. Metoda nejprve spustí metodu backtrack, čímž získá vektor souřadnic optimální cesty. Z tohoto vektoru jsou posléze vymazány všechny stavy, které nejsou skryté. Nalezené skryté stavy utváří návratové hodnoty této metody. Skryté stavy se na optimální cestě vyskytnou pouze v případě, pokud byla v předchozím kroku zavolána metoda search s pravdivým parametrem goPinchPoints.

Metoda getPinchPointsCoords je v rozhraní zdrojového kódu zapsána v následujícím tvaru:

```
queue<LPinchPoint> getPinchPointsCoords(HANDLE &stopRequest);
```

Metoda getPinchPointsCoords vyžaduje následující vstup:

- adresu handlu, který umožňuje uživateli backtracking přerušit (pracovního vlákna, ve kterém je algoritmus spuštěn).

Návratovým datovým typem metody backtrack je fronta souřadnic skrytých stavů, které se nacházejí na optimální cestě.

Metoda getStat

Metoda getStat vrací statistiky ve formě třídy LStat. Třída LStat obsahuje informace o daném prohledávání (počet expandovaných stavů, délku optimální cesty, dobu běhu plánovacího algoritmu apod.).

5.2.2 LState

Pomocí třídy LState se vytváří stavový prostor, ve kterém probíhá vyhledávání cesty pomocí metod třídy LBestFirstSearch. Každý stav obsahuje následující informace: index, index rodičovského stavu, vektor indexů potomků, vzdálenost do výchozího stavu, typ stavu, souřadnice ve stavovém prostoru, heuristické ohodnocení a další. Součástí LState je také metoda pro výpočet heuristického odhadu. Jako heuristický odhad se pro účely plánování pomocí algoritmu A* používá Euklidovská vzdálenost (Stanford Theory, 2015).

5.2.3 LFaceGraph

Třída LFaceGraph obsahuje metody jejichž cílem je vytvoření grafu sousedů. Inspirace pro vytvoření této třídy byla získána z práce (Ferguson, Stentz, 2004b). Graf sousedů je v této práci popsán v podkapitole 3.3.3 Graf sousedů. Graf sousedů se skládá z počátečního stavu, cílového stavu a skrytých stavů. Mezi všemi členy grafu sousedů jsou nalezeny nejkratší cesty (pokud existují). Při hledání těchto nejkratších cest se všechny skryté stavy dočasně uvažují jako překážky. Graf sousedů využívá třída LAOStar pro plánování v neurčitém prostředí pomocí metody AO*. Metoda AO* díky grafu sousedů redukuje prohledávání AND/OR grafu pouze na skryté stavy (plus výchozí a cílový stav). Při využití grafu sousedů dosáhnou metody AO* výsledného řešení rychleji, jak uvádí (Ferguson, Stentz, Thrun, 2004).

Jak jsem zmínil v předchozím textu, graf sousedů se skládá z počátečního, cílového stavu a skrytých stavů. V simulovaném prostředí se může vyskytovat mnoho skrytých stavů. Všechny tyto skryté stavy však nemusejí být nutně zahrnuty do grafu sousedů. Autoři algoritmu PAO* Ferguson, Stentz (2004b) provádějí před vytvořením grafu

sousedů akci nazvanou extrakce skrytých stavů. Díky této akci nemusejí být uvažovány všechny skryté stavy v prostředí. Algoritmus extrakce skrytých stavů je uveden níže (Algoritmus 6). Třída LFaceGraph má implementovánu extrakci skrytých stavů s následujícím rozdílem. Místo algoritmu D* využívám ve své implementaci pro hledání cest algoritmus A* (pomocí třídy LBestFirst). V tomto bodě implementace je vytvořen prostor pro případné vylepšení extrakce skrytých stavů, tak jak ji obsahuje třída LFaceGraph. Jak uvádí Ferguson a Stentz (2004b), pro danou aplikaci překonává algoritmus D* algoritmus A*.

Třída LFaceGraph používá stavy, které jsou utvořeny pomocí třídy LStateFace. Jedná se o obdobu třídy LState, avšak speciálně přizpůsobenou pro potřeby vytváření grafu sousedů. Dále je v LFaceGraf použita vytvořená třída LPinchPoint. LPinchPoint je třída, která obsahuje souřadnice skrytého stavu a souřadnice buňky, kterou předchází skrytý stav na cestě. Využití třídy LPinchPoint se nachází při extrakci skrytých stavů.

1) Na počátku se v prostředí nastaví dočasně všechny buňky na typ průchozí. Pomocí algoritmu D* se vytvoří plán optimální cesty ze startu do cíle. Pokud se na této cestě vyskytují skryté stavy, zaznamenají se do fronty *F*.
2) Pokud fronta *F* není prázdná proveď následující pod-kroky:
   a) *Nalezni novou cesty ze skrytého stavu do cíle*: Odstraň první prvek (skrytý stav) z fronty a nastav buňku, která přísluší skrytému stavu na typ neprůchozí (překážka). Spusť algoritmus D*, který pře-plánuje cestu do cíle z buňky, která předcházela na cestě skrytému stavu.
   b) *Zaznamenej skryté stavy na nové cestě*: Přidej všechny skryté stavy na nové cestě na konec fronty *F*.
3) Předej jako návratovou hodnotu vektor zaznamenaných skrytých stavů.

Algoritmus 6: Extrakce skrytých stavů (Ferguson, Stentz, 2004b)

5.2.4 LAOStar

Třída LAOStar obsahuje metody, které prohledávají stavový prostor pomocí algoritmu AO*, případně WAO*. Před spuštěním algoritmu AO* se uplatňují metody třídy LFaceGraph, které vytvoří graf sousedů, nad kterým je následně spuštěn algoritmus AO* / WAO*. Při implementaci algoritmu AO* jsem vycházel z literatury (Aksakalli, 2007b; Březina, Dvořák, 2013a; Ferguson, Stentz, Thrun, 2004).

Metoda search

Metoda search třídy LAOStar slouží k nalezení předpokládané optimální cesty v neurčitém prostředí. Pro plánování cesty využívá tato metoda algoritmus AO* (3.2 Algoritmus AO*). Implementace algoritmu AO* ve třídě LAOStar má zavedenu proměnnou typu double – weight (váha). Parametr weight může nabývat hodnoty větší nebo rovné jedné. V případě, že je tento parametr roven jedné, pak metoda search nalezne optimální řešení. Změnou parametru weight (na hodnotu větší jak jedna) dojde k modifikaci algoritmu AO* na algoritmus WAO* (3.6.1 Algoritmus WAO*). Algoritmus WAO* nemusí vždy nalézt optimální řešení (Chung, Huang, 2011). Model neurčitého prostředí, který metoda search prohledává je popsán v 4.2 Model simulačního prostředí.

Metoda search je v rozhraní zdrojového kódu zapsána v následujícím tvaru:

```
int search(LCoords sSource, LCoords sTarget, double disambiguationCost,
           int futility, HANDLE &stopRequest, HWND hWnd, UINT message,
           double weight, LFaceGraph* _faceGraph);
```

Metoda search vyžaduje následující vstupy:

- souřadnice startu a cíle (sSource, sTarget),
- cenu zjednoznačnění skrytého stavu (disambiguationCost),
- maximální přípustnou cenu řešení (futility),
- adresu handlu (&stopRequest), který umožňuje uživateli metodu search přerušit,
- Handle zařízení (hWnd), na které budou zasílány zprávy o průběhu prohledávání stavového prostoru (message),
- parametr weight, který ovlivňuje heuristiku (3.6.1 Algoritmus WAO*),
- graf sousedů (faceGraph), který byl vytvořen před zavoláním metody search.

Metoda search vrací hodnotu typu int, která v daném případě vyjadřuje délku nalezené cesty. V případě, že cesta nebyla nalezena, je návratová hodnota rovna nule.

Metoda getPath

Metoda getPath slouží k vytvoření vektoru souřadnic optimální cesty. Tuto metodu je vhodné volat až po metodě search, která nalezne cestu ze startu do cíle (pokud existuje). Metoda getPath začne hledat souřadnice optimální cesty od kořenového vrcholu. Poté následuje nejnadějnější podstromy, které tvoří dané řešení.

Metoda getPath je v rozhraní zdrojového kódu zapsána v následujícím tvaru:

```
vector<LCoords> getPath(HANDLE &stopRequest);
```

Metoda getPath vyžaduje následující vstup:

- adresu handlu (&stopRequest), který umožní uživateli metodu getPath přerušit před jejím řádným ukončením.

Metoda getPath vrací vektor souřadnic, které dohromady utvářejí plán optimální cesty. V případě, že cesta neexistuje, pak uvedená metoda vrátí prázdný vektor souřadnic.

Metoda getStat

Třída LAOStar obsahuje dále veřejnou metodu getStat, která je shodná s metodou getStat uvedenou v 5.2.1 LBestFirstSearch. Metoda getStat vrací informace o proběhlém prohledávání stavového prostoru pomocí třídy LStat.

5.2.5 LState2

Třída LState2 je obdobou LState, avšak slouží k vytváření stavového prostoru pro metody AO* (implementované ve třídě LAOStar). Třída LState2 obsahuje následující rozšíření: ukazatel na nejlepšího potomka, vektor souřadnic cesty do rodičovského vrcholu a pravděpodobnost překážky skrytého stavu.

6 OVĚŘOVACÍ EXPERIMENTY

Cílem této práce je provést v implementovaném simulačním prostředí ověřovací experiment. Jako ověřovací experiment jsem zvolil test, při které se bude sledovat vliv nastavení váhy algoritmu WAO* na optimalitu výsledného řešení. Testy (včetně jejich cílů) jsou popsány v příslušných podkapitolách (6.1 Test hledání optimality algoritmu WAO* a 6.2 Ověřovací test optimality algoritmu WAO*). Inspiraci pro provedení experimentu s WAO* jsem nalezl v literatuře (Chung, Huang, 2011). Autoři Chung a Huang (2011) rozšiřují algoritmus WAO* na algoritmus DAO* (viz podkapitola 3.6.1 Algoritmus WAO*). Úpravou váhy heuristické funkce lze docílit rychlejšího nalezení řešení (Chung, Huang, 2011). Řešení, které je v takovém případě nalezeno, nemusí být optimální. V praxi není vždy nutné nalézt optimální cestu. Hledání optimální cesty může být pro plánovací algoritmus náročné na čas. Robot však v reálných podmínkách může potřebovat zareagovat okamžitě a na další výpočty cesty nemá čas. V takovém případě je výhodné mít k dispozici algoritmus, který je schopen poskytnout rychle i když neoptimální řešení.

Všechny experimenty v této kapitole byly provedeny na osobním počítači s procesorem i3-4150 a se 4 GB operační paměti. Simulátor byl spuštěn na 64-bitovém operačním systému Windows 8.1. Program AO sim byl zkompilován pro 32-bitovou i 64-bitovou verzi systému Windows. Všechny následující experimenty byly provedeny v simulátoru, který byl zkompilován pro architekturu procesorů x64.

Všechny experimenty, které jsou zahrnuty v této práci, se nacházejí na přiloženém CD. K dispozici jsou soubory typu .gld, které lze otevřít v aplikaci AO sim. Zároveň jsou na CD přítomny csv soubory, které obsahují statistiky z proběhlých simulacích. K provedeným experimentům je zapotřebí dodat následující komentář. Všechny experimenty byly provedeny ve verzi aplikace AO sim, ve které byla zanesena chyba způsobující neukládání cesty algoritmu WAO*. Tato chyba nemá vliv na statistiky provedených experimentů. Pro zobrazení cesty algoritmu WAO* v přiložených souborech je nutné provést nové plánování cesty (v aplikaci menu zvolit Run → WAO* a zároveň nastavit váhu na příslušnou hodnotu). Chyba nemá vliv na soubory u nichž je cesta algoritmu AO* a WAO* shodná. Popsaná chyba byla z finální verze programu AO sim odstraněna.

6.1 Test hledání optimality algoritmu WAO*

Cílem testu optimality algoritmu WAO* je nalézt nejvyšší hodnotu váhy (koeficientu nadhodnocení heuristické hodnoty pro algoritmus WAO*), při které ještě je nalezena optimální cesta pro dané parametry prostředí (rozměr, počet překážek). Nalezenou nejvyšší hodnotu váhy by bylo možné uplatnit u anytime metod AO*. Autoři pře-plánovací metody DAO* Chung a Huang (2011) volí, v případě zmíněné metody, váhu mezi hodnotami 3 až 10.

K provedení experimentu jsem využil implementovaný program AO sim. Automatický test v prvním kroku vygeneroval náhodnou mapu dle příslušného nastavení. Poté byly postupně spuštěny následují plánovací algoritmy: A*, AO* a WAO*. Algoritmy A* a AO* byly spuštěny pro každé prostředí pouze jednou. U algoritmu WAO* se však postupovalo následovně. Nejprve se nastavila váha WAO* na hodnotu 1. Hodnota váhy byla následně navýšena o krok uvedený v nastavení automatického testu. Následně bylo provedeno plánování pomocí algoritmu WAO*. Pokud byla nalezená cesta pomocí algoritmu WAO* shodná s cestou nalezenou pomocí AO*, pak se inkrementovala hodnota váhy o daný krok a algoritmus WAO* se spustil znovu. Tento postup skončil ve chvíli

nalezení neoptimální cesty algoritmem WAO*. Do statistik byla zaznamenána hodnota váhy, která ještě zajistila optimální cestu pro dané prostředí.

6.1.1 Test prostředí 83x33

Testovací nástroj byl nastaven následujícím způsobem:

- Rozměry prostředí: 83x33
- Počet náhodně generovaných skrytých stavů: 300
- Počet transformací dočasných značek: 0
- Náhodné souřadnice startu a cíle: ne
- Generování skrytých stavů pouze do překážek: ano
- Náhodná pravděpodobnost skrytých stavů: ano
- Krok váhy algoritmu WAO*: 0.1
- Počet testů: 63
- Souřadnice výchozí polohy robota: [40, 1]
- Souřadnice cílové polohy robota: [40, 31]
- Cena zjednoznačnění: 1

Tabulka 2: Test prostředí 83x33

	A*	AO*	WAO*
Délka cesty			
Min	86	68	68
Max	466	394	394
Průměr	307,0	213,5	213,5
Počet expandovaných stavů			
Min	482	1030	325
Max	1307	169567	77924
Průměr	1138	34161	5690
Doba běhu algoritmu (s)			
Min	0,001	0,485	0,141
Max	0,012	228,588	145,191
Průměr	0,006	41,367	7,318
Počet skrytých stavů na cestě			
Min	0,0	1,0	1,0
Max	0,0	4,0	4,0
Průměr	0,0	1,7	1,7

Tabulka 3: Test prostředí 83x33 – doplňují informace

	Počet uvažovaných stavů (-)	Čas extrakce skrytých stavů (s)	Doba zametání (s)
Min	123	0,2	7,4
Max	264	1,6	159,1
Průměr	214	0,8	74,5

Tabulka 4: Test prostředí 83x33 – statistiky váhy

	WAO* - váha (-)
Min	1,6
Max	8,9
Průměr	4,5

V popsaném experimentu bylo provedeno celkem 63 testů. Průměrné nastavení váhy algoritmu WAO*, při které ještě bylo nalezeno optimální řešení je hodnota 4,5. Maximální a minimální nastavení váhy, pří kterém byla nalezena optimální cesta jsou shrnuty v tabulce (Tabulka 4: Test prostředí 83x33 – statistiky váhy). Získanou minimální hodnotu váhy jsem využil pro provedení ověřovacího experimentu v podkapitole 6.2.1 Test prostředí 83x33 – konstantní váha 1,6. Z ověřovacího experimentu podkapitoly 6.2.1 je možné vysledovat počet neoptimálních cest při využití algoritmu WAO* s váhou nastavenou na hodnotu 1,6 (minimální váhu z experimentu v této podkapitole).

6.1.2 Test prostředí 43x17

Testovací nástroj byl nastaven následujícím způsobem:

- Rozměry prostředí: 43x17
- Počet náhodně generovaných skrytých stavů: 150
- Počet transformací dočasných značek: 0
- Náhodné souřadnice startu a cíle: ne
- Generování skrytých stavů pouze do překážek: ano
- Náhodná pravděpodobnost skrytých stavů: ano
- Krok váhy algoritmu WAO*: 0.1
- Počet testů: 4016
- Souřadnice výchozí polohy robota: [21, 1]
- Souřadnice cílové polohy robota: [21, 15]
- Cena zjednoznačnění: 1

Tabulka 5: Test prostředí 47x17

	A*	AO*	WAO*
Délka cesty			
Min	18	14	14
Max	214	142	142
Průměr	94,8	52,2	52,2
Počet expandovaných stavů			
Min	28	25	23
Max	335	108308	70671
Průměr	257	4357	1495
Doba běhu algoritmu (s)			
Min	0,000	0,000	0,000
Max	0,008	28,720	19,584
Průměr	0,001	0,619	0,219
Počet skrytých stavů na cestě			
Min	0,0	1,0	1,0
Max	0,0	6,0	6,0
Průměr	0,0	1,6	1,6

Tabulka 6: Test prostředí 47x17 – doplňují informace

	Počet uvažovaných stavů (-)	Čas extrakce skrytých stavů (s)	Doba zametání (s)
Min	11	0,000	0,0
Max	151	0,118	5,5
Průměr	108	0,038	1,7

Tabulka 7: Test prostředí 47x17 – statistiky váhy

	WAO* - váha (-)
Min	1,2
Max	45,1
Průměr	3,7

V tomto experimentu bylo vygenerováno 4016 testů. Rozměr prostředí pro tento experiment byl zvolen o velikost 47x17. Díky rozměru prostředí, který je menší než v případě experimentu v podkapitole 6.1.1.Test prostředí 83x33, mohlo proběhnout testů více. Hodnota váhy algoritmu WAO*, při které ještě bylo nalezeno optimální řešení je v tomto experimentu průměrně rovna 3,9. Maximální a minimální nastavení váhy, při které byla nalezena optimální cesta jsou shrnuty v tabulce (Tabulka 7: Test prostředí 47x17 – statistiky váhy). Získanou minimální hodnotu váhy jsem využil pro provedení ověřovacího experimentu v podkapitole 6.2.2 Test prostředí 47x17 – konstantní váha 1,2. Z ověřovacího experimentu podkapitoly 6.2.2 je možné vysledovat počet neoptimálních cest při využití algoritmu WAO* s váhou nastavenou na hodnotu 1,2 (minimální váhu z experimentu v této podkapitole).

6.2 Ověřovací test optimality algoritmu WAO*

Cílem ověřovacího testu optimality WAO* je zjistit, v kolika případech nalezené řešení nebude optimální. V této podkapitole byly provedeny dva testy. První test byl spuštěn v prostředí 83x33 s konstantní váhou 1,6. Druhý test byl spuštěn v prostředí 47x17 s váhou 1,2. Uvedené váhy byly získány experimentem v podkapitole 6.2 Ověřovací test optimality algoritmu WAO*.

K provedení experimentu byl využit nástroj pro automatické testování, který se nachází v programu AO sim. Testovací nástroj postupně generoval náhodná prostředí o velikosti 83x33. Konkrétní nastavení testovacího nástroje je uvedeno v následujících podkapitolách. Testovací nástroj spustil pro každé vygenerované prostředí postupně algoritmy: A*, AO* a WAO* s váhou 1,6.

6.2.1 Test prostředí 83x33 – konstantní váha 1,6

Testovací nástroj byl nastaven následujícím způsobem:

- Rozměry prostředí: 83x33
- Počet náhodně generovaných skrytých stavů: 300
- Počet transformací dočasných značek: 0
- Náhodné souřadnice startu a cíle: ne
- Generování skrytých stavů pouze do překážek: ano
- Náhodná pravděpodobnost skrytých stavů: ano
- Krok váhy algoritmu WAO*: 0
- Počet testů: 198
- Souřadnice výchozí polohy robota: [40, 1]
- Souřadnice cílové polohy robota: [40, 31]
- Váha WAO*: 1,6
- Cena zjednoznačnění: 1

Tabulka 8: Test prostředí 83x33

	A*	AO*	WAO*
Délka cesty			
Min	106	38,0	38,0
Max	546	506,0	506,0
Průměr	299	193,6	194,0
Počet expandovaných stavů			
Min	380	445	225
Max	1309	691835	497700
Průměr	1114	33758	25218
Doba běhu algoritmu (s)			
Min	0,001	0,342	0,196
Max	0,012	1402,070	1055,305
Průměr	0,007	48,599	35,281
Počet skrytých stavů na cestě			
Min	0,0	1,0	0,0
Max	0,0	7,0	7,0
Průměr	0,0	1,8	1,8

Tabulka 9: Test prostředí 83x33 – doplňují informace

	Počet uvažovaných stavů (-)	Čas extrakce skrytých stavů (s)	Doba zametání (s)
Min	105	0,2	7,4
Max	283	2,0	198,8
Průměr	214	0,8	83,2

V uvedeném experimentu bylo provedeno 198 testů. U těchto 198 testů nebylo nalezeno optimální řešení v případě algoritmu WAO* ve 4 případech. Procentuálně vyjádřeno bylo v uvedené případu u algoritmu WAO* neoptimálních celkem 2,02 % testů.

V následujícím textu budou krátce okomentovány jednotlivé případy prostředí, u kterých algoritmus WAO* nenalezl optimální řešení. Ne-optimalita naplánované cesty pomocí WAO* může mít (dle tohoto experimentu) za následek následující:

- Nalezená cesta bude kratší než cesta nalezená pomocí AO*, ale na nalezené neoptimální cestě se bude vyskytovat více skrytých stavů, které zvýší pravděpodobnost výskytu neurčité překážky na cestě. Tuto situaci dokládá prostředí uložené v souboru „test150507_A_weight_check5.gld“. Cesta nalezená pomocí algoritmu WAO* je o 62 kroků kratší, avšak na dané cestě se nachází dva skryté stavy navíc. V případě cesty nalezené pomocí AO* se na dané cestě nachází pouze jeden skrytý stav.
- Nalezená cesta bude delší než cesta nalezená pomocí AO* a zároveň se na této cestě bude vyskytovat méně skrytých stavů. Konkrétně tuto situaci dokládá prostředí uložené v souboru „test150507_A_weight_check16.gld“.Vynechání jednoho skrytého stavu se v daném případě nevyplatí (pravděpodobnost překážky daného stavu je 10%). Délka cesty se v tomto případě prodlouží o 40 kroků.
- Nalezená cesta bude delší než cesta nalezená pomocí algoritmu AO* a zároveň se na této cestě nebude nacházet žádný skrytý stav. Tuto situaci dokládá soubor „test150507_A_weight_check29.gld“.

- Cesta nalezená pomocí algoritmu WAO* bude delší než cesta nalezená pomocí AO* a zároveň obě cesty budou obsahovat obsahovat stejný počet skrytých stavů. Přičemž pravděpodobnosti, že dané skryté stavy mohou být překážkou budou shodné v obou případech. Tento případ dokládá soubor „test150507_A_weight_check58.gld“.

6.2.2 Test prostředí 47x17 – konstantní váha 1,2

Testovací nástroj byl nastaven následujícím způsobem:

- Rozměry prostředí: 43x17
- Počet náhodně generovaných skrytých stavů: 150
- Počet transformací dočasných značek: 0
- Náhodné souřadnice startu a cíle: ne
- Generování skrytých stavů pouze do překážek: ano
- Náhodná pravděpodobnost skrytých stavů: ano
- Krok váhy algoritmu WAO*: 0
- Počet testů: 10556
- Souřadnice výchozí polohy robota: [21, 1]
- Souřadnice cílové polohy robota: [21, 15]
- Váha WAO*: 1,2
- Cena zjednoznačnění: 1

Tabulka 10: Test prostředí 43x17

	A*	AO*	WAO*
Délka cesty			
Min	18	14	14
Max	254	164	164
Průměr	90,9	50,4	50,7
Počet expandovaných stavů			
Min	23	11	7
Max	335	187348	187553
Průměr	248	3971	3616
Doba běhu algoritmu (s)			
Min	0,000	0,000	0,000
Max	0,002	38,785	39,929
Průměr	0,001	0,634	0,565
Počet skrytých stavů na cestě			
Min	0,0	1,0	0,0
Max	0,0	7,0	7,0
Průměr	0,0	1,5	1,5

Tabulka 11: Test prostředí 43x17 – doplňují informace

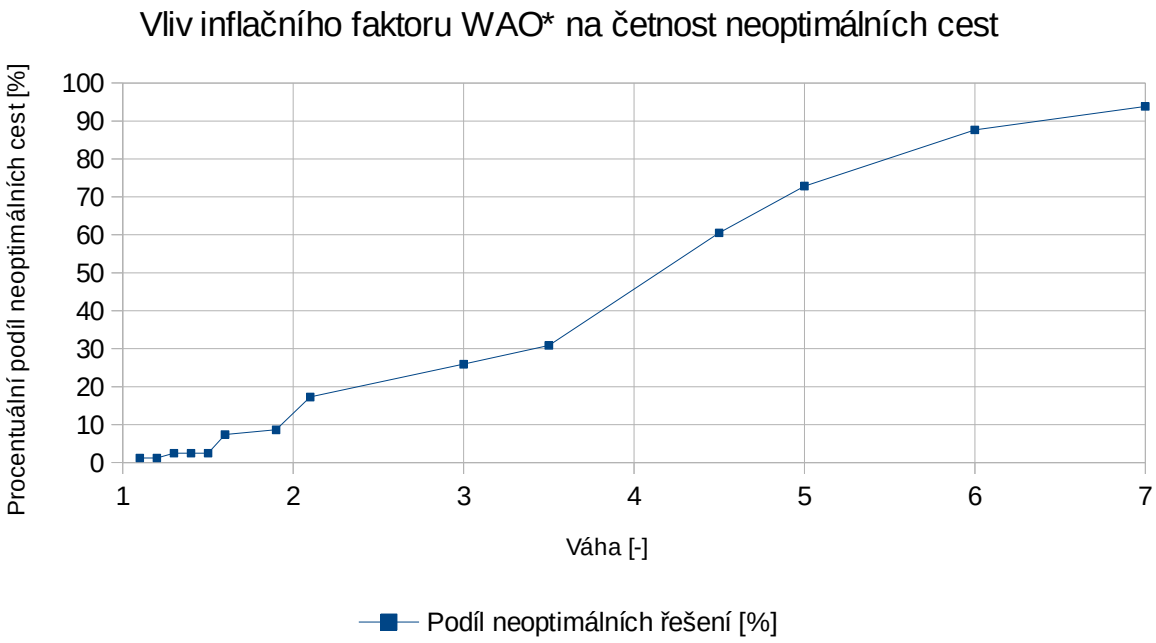
	Počet uvažovaných stavů (-)	Čas extrakce skrytých stavů (s)	Doba zametání (s)
Min	4	0,000	0,0
Max	152	0,127	6,1
Průměr	103	0,037	1,8

V uvedeném experimentu bylo provedeno 10559 testů. U těchto 10559 testů nebylo nalezeno optimální řešení v případě algoritmu WAO* v 337 případech. Procentuálně vyjádřeno bylo v uvedené případu u algoritmu WAO* neoptimálních celkem 3,02 % testů.

6.3 Vliv inflačního faktoru

Pro zjištění vlivu inflačního faktoru algoritmu WAO* na četnost neoptimálních cest jsem provedl následující experiment. Pro prostředí o velikosti 83x33 jsem vygeneroval pomocí programu AO sim 81 testů zvlášť pro inflační faktor: 1,1 až 7. Ve vygenerovaných prostředích se vždy vyskytovalo 300 náhodně rozmístěných stavů. Skryté stavy byly generovány pouze na místo překážek (bylo potvrzeno nastavení testovacího nástroje „*Numbers of base → obstacle transformation*“). Na naplánované cestě se vždy vyskytoval alespoň jeden skrytý stav, naopak nejvýše 7 skrytých stavů. Vliv inflačního faktoru na optimalitu výsledného řešení je zobrazen na grafu na obrázku (Obr. 22: Vliv inflačního faktoru algoritmu WAO* na četnost neoptimálních řešení).

Při nastavení váhy algoritmu WAO* na hodnoty 1,1 a 1,2 bylo vygenerováno 1,2 % neoptimálních cest. Mírné stoupnutí počtu neoptimálních řešení nastalo u nastavení váhy na hodnoty 1,3 až 1,5. Při uvedeném nastavení bylo nalezeno 2,4 % neoptimálních cest. S dalším zvyšováním váhy začíná však počet neoptimálních cest růst rychleji. Pokud byla váha nastavena na hodnotu 7, pak počet neoptimálních řešení byl 93,8 % z celku.

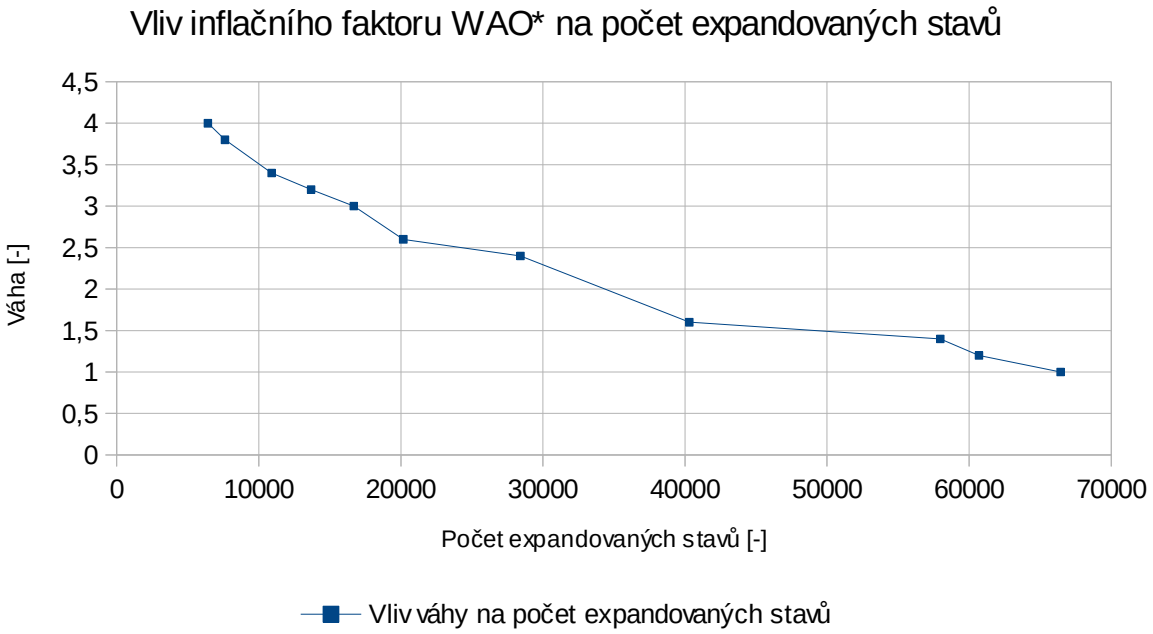


Obr. 22: Vliv inflačního faktoru algoritmu WAO na četnost neoptimálních řešení*

6.4 Test prostředí

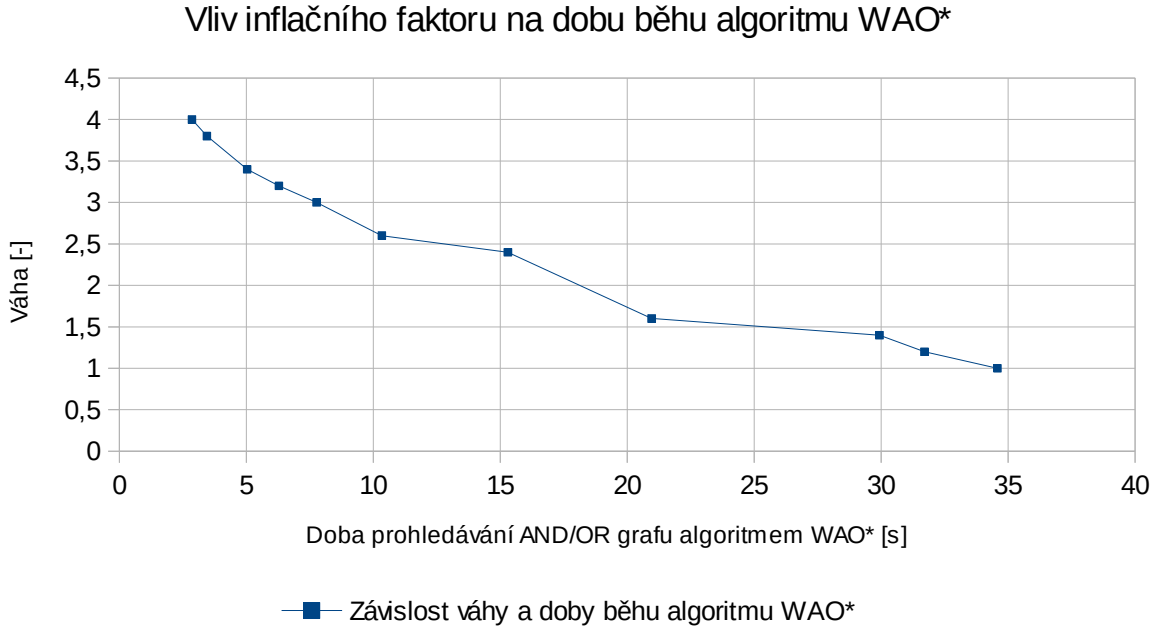
V této podkapitole se zaměřím na sledování vlivu inflačního faktoru algoritmu WAO* na počet expandovaných stavů a na dobu řešení. Pro provedení tohoto experimentu jsem vytvořil testovací prostředí (43x17) pomocí programu AO sim. Do prostředí jsem rozmístil náhodně 150 skrytých stavů. U testovaného prostředí jsem nejdříve nastavil váhu algoritmu WAO* na hodnotu jedna. Následně jsem spustil algoritmus WAO* nad daným prostředím. Vygenerované statistiky jsem zapsal do souboru. Poté jsem navýšil hodnotu inflačního faktoru o hodnotu 0,2 a celý proces jsem provedl znovu. Výsledky provedeného experimentu jsou uvedeny v grafech níže. Pro tyto testy jsem využil počítač s procesorem Intel Celeron 2957U (1,4GHz) a s 4 GB paměti. Program byl spuštěn na 64-bitovém OS Windows 8.1.

6.4.1 Test č.1



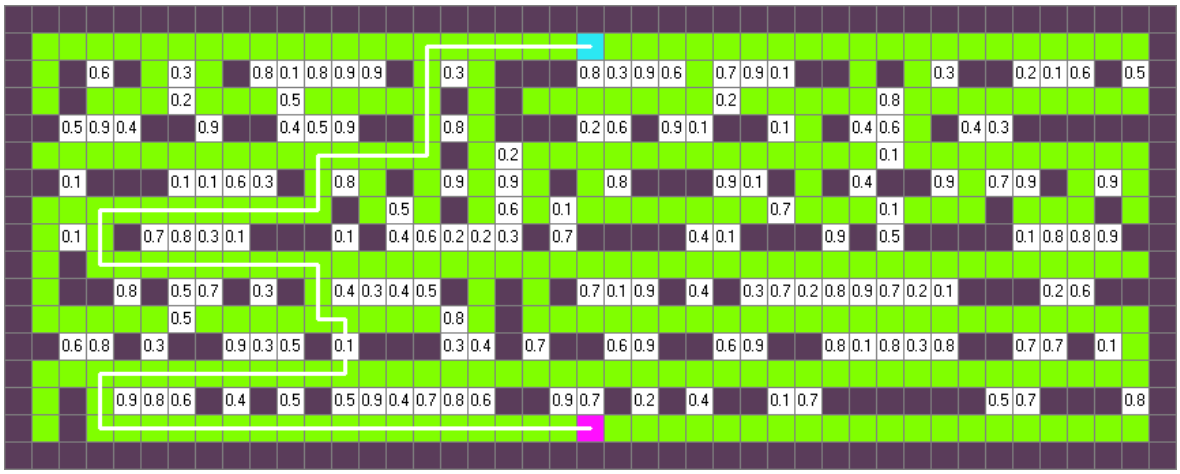
Obr. 23: Vliv inflačního faktoru WAO na počet expandovaných stavů*

Graf na obrázku (Obr. 23) zobrazuje závislost vlivu nastavení váhy WAO* na počet expandovaných stavů. S rostoucí hodnotou váhy, klesá počet expandovaných stavů. Nejvyšší hodnota váhy, která ještě zaručí nalezení optimálního řešení, je v tomto testu rovna 2,6. Algoritmus AO* expandoval 66439 stavů, algoritmus WAO* s váhou 2,6 expandoval 20173 stavů. Algoritmus WAO* s váhou 2,6 potřeboval k nalezení řešení expandovat o 70 % méně stavů, než algoritmus AO*.



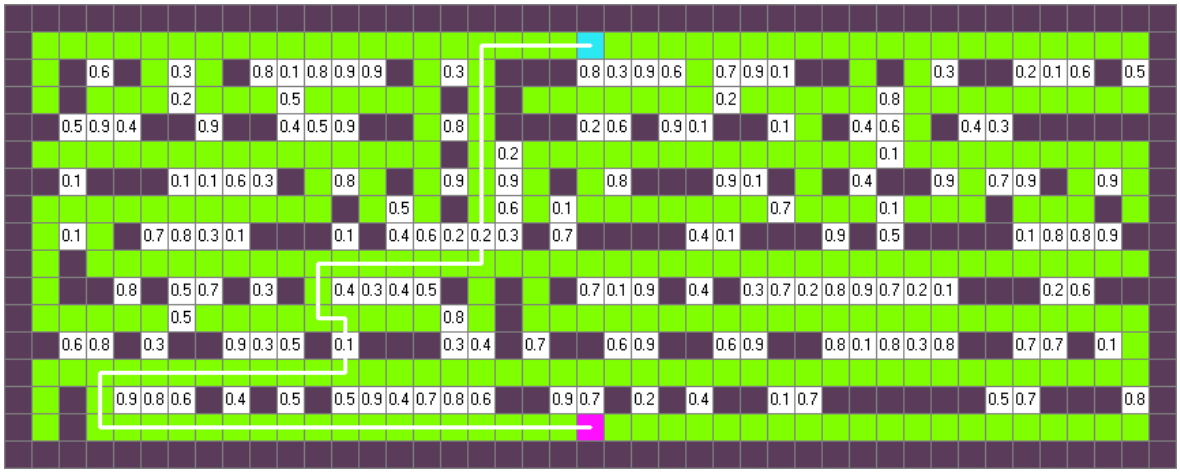
*Obr. 24: Vliv inflačního faktoru na dobu běhu algoritmu WAO**

Graf na obrázku (Obr. 24) zobrazuje vliv nastavení váhy WAO* na dobu provedení algoritmu. S rostoucí hodnotou váhy, klesá doba potřebná pro nalezení řešení. Nejvyšší hodnota váhy, která je stále ještě schopna nalézt optimálního řešení, je v tomto testu rovna 2,6. Algoritmus AO* potřeboval k nalezení řešení dobu 35,69s. V případě algoritmu WAO* s váhou 2,6 bylo řešení nalezeno za 10,34s. Algoritmus WAO* s váhou 2,6 je v daném případě o 72 % rychlejší než algoritmus AO*.



Obr. 25: Očekávané optimální řešení v prostředí 43x17 pomocí AO a WAO* (váha: 2,6); délka cesty: 68; počet skrytých stavů na cestě: 1*

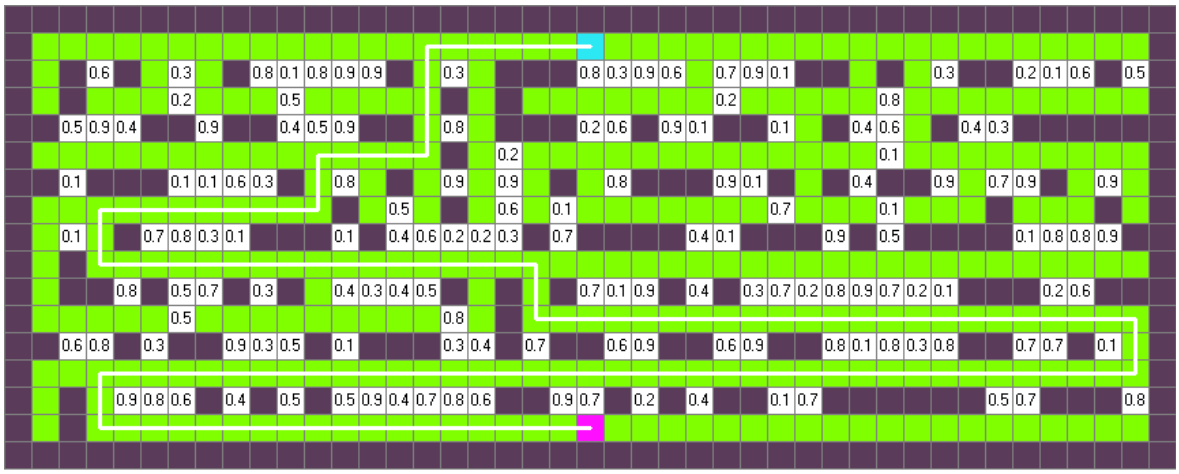
Na obrázku (Obr. 25) je testované prostředí, ve kterém je vyznačena očekávaná optimální cesta. Tuto cestu nalezly algoritmy AO* a WAO* s váhou 2,6. Na dané cestě se vyskytuje pouze jeden neurčitý stav. Uvedený neurčitý stav má přiřazenu pravděpodobnost překážky rovnu 0,1.



Obr. 26: Neoptimální řešení WAO (váha: 2,7); délka cesty: 52; počet skrytých stavů na cestě: 2*

V případě zvýšení váhy na hodnotu 2,7 došlo ke změně výsledku, řešení již není optimální. Tento případ je zobrazen na obrázku (Obr. 26). Nalezená cesta se skládá z 52

polí. Tato cesta je tedy kratší, než v případě cesty získané s algoritmem AO* nebo WAO* s váhou 2,6 (Obr. 25). Nicméně na cestě nalezené algoritmem WAO* s váhou 2,7 se vyskytují dva neurčité stavy. Tyto neurčité stavy zvyšují pravděpodobnost, že se na naplánované cestě vyskytne překážka.



Obr. 27: Cesta nalezená algoritmem A; délka cesty: 126*

Cesta, kterou naplánoval algoritmus A* pro testované prostředí, je zobrazena na obrázku výše (Obr. 27). Uvedená cesta se skládá ze 126 buněk. Plán vytvořený pomocí algoritmu A* nezahrnuje uvažování neurčitostí. Tato cesta je o 58 buněk delší než v případě cesty naplánované algoritmem AO*, který zahrnuje do plánu neurčitosti prostředí.

6.5 Shrnutí testů

Testy provedené v těchto podkapitolách 6.3 Vliv inflačního faktoru a 6.4 Test prostředí dokládají možnosti algoritmu WAO*. Algoritmus WAO* je možné použít jako základ pro další rozšíření metod AO*. Jedná se zejména o rozšíření na přeplánovací algoritmy viz algoritmy DAO* a DDAO* (Chung, Huang (2011)). Přeplánovací algoritmus může nastavit počáteční váhu na vysoké číslo. V provedeného testu (Obr. 22: Vliv inflačního faktoru algoritmu WAO* na četnost neoptimálních řešení) byla nejvyšší hodnota váhy, která umožnila nalezení optimální řešení, rovna 7. Algoritmus WAO* s váhou nastavenou na hodnotu 7, nalezl řešení v průměru za 1,28s (AO* nalezl řešení v průměru za 34,33s) pro prostředí testované v 6.3 Vliv inflačního faktoru. Takto nastavený algoritmus WAO* vyhledává poměrně mnoho neoptimálních řešení(Obr. 22: Vliv inflačního faktoru algoritmu WAO* na četnost neoptimálních řešení). V reálných podmínkách při navigaci robota může být kladen důraz na rychlé poskytnutí řešení, které nemusí být optimální. Toto neoptimální řešení se však může postupně vylepšovat snižováním váhy, pokud je k dispozici čas na další plánování (viz 3.6 Anytime algoritmy a nejisté dynamické prostředí).

7 ZÁVĚR

Cíle této diplomové práce, tak jak byly zadány, byly splněny v požadovaném rozsahu. V teoretické části práce jsem analyzoval metody AO* určené pro plánování cesty v neurčitém prostředí. Praktická část práce se zaměřila na implementaci vybraných metod AO*. V rámci praktické části práce bylo vytvořeno simulační prostředí, které slouží k provedení experimentů s implementovanými metodami AO*.

V praktické části této práce jsem se zaměřil na model, který předpokládá plánování cesty v neurčité mapě. Modelový příklad předpokládá, že robot dostane k dispozici mapu prostředí. Mapa prostředí může být pořízena například bezpilotním letounem, či satelitem. V pořízené mapě se mohou vyskytovat nejasnosti, které jsou následkem nízkého rozlišení mapy. K vybraným neurčitým místům lze přiřadit pravděpodobnost, že jsou neprůchozí. Cílem plánování v neurčité mapě je poskytnout strategii, která umožní robotovi projít prostředím ze startu do cíle s účelem minimalizovat očekávanou délku cesty. Nalezení optimálního řešení daného problému je možné pomocí algoritmu AO*.

V této práci jsem implementoval metodu AO* určenou pro plánování v prostředí s neurčitou mapou. Algoritmus AO* vždy nalezne optimální řešení. Uvedenou metodu AO* jsem rozšířil na algoritmus WAO*, který používá váženou heuristiku. Použitím vážené heuristiky lze docílit rychlejšího nalezení řešení. Řešení nalezené s využitím WAO* však nemusí být optimální. Přesto, že algoritmus WAO* nezaručuje nalezení optimálního řešení, jedná se o důležitý algoritmus pro praktické aplikace. V praktických podmínkách může být upřednostněna rychlost nalezení řešení před jeho optimalitou. Algoritmus WAO* je podkladem pro další rozšiřování metod AO*, zejména pak na metody typu anytime. Anytime metody jsou schopné velmi rychle nalézt řešení, které však nemusí být optimální. Avšak pokud má plánovací algoritmus k dispozici čas na další výpočty, pak postupně nalezené řešení vylepšuje. Pokud mají anytime metody k dispozici dostatek času, pak naleznou optimální řešení.

Pro provedení ověřovacích experimentů jsem v rámci této práce vytvořil simulační program. Prostředí je simulováno 2D mřížkou, ve které každá buňka představuje možnou polohu robota. Velikost jednotlivých buněk je dána staticky. Jemnost mřížky tedy nelze volit. Tato vlastnost programu však nebrání v provedení testů. Možnost zavedení volby jemnosti 2D mřížky je jedno z možných vylepšení stávajícího programu. Velikost prostředí (počet buněk) si volí uživatel. Robot se v simulačním prostředí může pohybovat v osmi směrech. Simulační program má k dispozici nástroje, které umožní umístit do prostředí překážky (určité i neurčité) a výchozí a cílovou polohu robota. Program umožňuje ruční i automatické vytváření prostředí. Vytvořené prostředí je možno uložit na disk počítače. Aby bylo možné provést ověřující experimenty, simulační program obsahuje nástroj, který si vede statistiky o aktuálním prostředí a proběhlých vyhledáváních. Statistiky je možné ukládat na disk počítače.

Provedené experimenty porovnávají algoritmy AO* a WAO*. Algoritmus WAO* poskytl v testovaném prostředí, kde byl nastaven vysoký inflační faktor (7 a více), řešení až o 99,56 % rychleji než algoritmus AO*. Je nutné vzít v úvahu, že při vysokém inflačním faktoru (7 a více) byl zjištěn 93,8 %-ní podíl neoptimálních cest. Naopak při mírném zvýšení inflačního faktoru (na hodnoty 1,1 až 1,2) bylo nalezeno 1,2 % neoptimálních cest. V daném případě dochází ke zkrácení času prohledávání o 8,9 %. V případě rozšíření algoritmu WAO* na metodu typu anytime, lze doporučit nastavení počátečního inflačního faktoru na hodnotu 7 a více. Při těchto hodnotách dochází k rychlému nalezení, obvykle neoptimálního, řešení. Řešení mohou metody anytime následně vylepšovat, pokud mají k dispozici potřebný čas.

SEZNAM POUŽITÉ LITERATURY

ABBADI, Ahmad a Radomil MATOUSEK. 2014. PATH PLANNING IMPLEM ENTATION USING MATLA B. *International Conference of Technical Computing* [online]. : 1-6 [cit. 2015-05-13]. DOI: 10.13140/2.1.3324.5767. Dostupné z: http://www.researchgate.net/publication/268980175_PATH_PLANNING_IMPLEMEN TATION_USING_MATLAB

AKSAKALLI, Vural. The BAO* algorithm for stochastic shortest path problems with dynamic learning. *Decision and Control: IEEE Conference* [online]. 2007, (46) [cit. 2015-01-31]. DOI: 10.1109/CDC.2007.4434409. Dostupné z: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4434409

AKSAKALLI, Vural. 2007b. *Protocols for Stochastic Shortest Path Problems with Dynamic Learning* [online]. Baltimore, Maryland [cit. 2015-05-08]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download? doi=10.1.1.110.8955&rep=rep1&type=pdf. Dissertation. The Johns Hopkins University.

BANDER, James L. a Chelsea C. WHITE,. 2002. A Heuristic Search Approach for a Nonstationary Stochastic Shortest Path Problem with Terminal Cost. *Transportation Science* [online]. (36): 218–230 [cit. 2015-05-08].

BERG, Jur van den, Dave FERGUSON a James KUFFNER. 2006. *Anytime Path Planning and Replanning in Dynamic Environments* [online]. [cit. 2015-05-09]. Dostupné z: http://www.ri.cmu.edu/pub_files/pub4/berg_jur_van_den_2006_1/berg_jur_van_den_2 006_1.pdf

BENAZERA, Emmanuel, Eric A. HANSEN, Ronen BRAFMA a Nicolas MEULEA. An AO* Algorithm for Planning with Continuous Resources. *Dept. of Computer Science and Engineering* [online]. 2005 [cit. 2015-02-22]. Dostupné z: http://ti.arc.nasa.gov/m/pub-archive/973h/0973%20%28Benazera%29.pdf

BŘEZINA, Tomáš a Jiří DVOŘÁK. *Algoritmy umělé inteligence: Lekce 3*. 2013a.

BŘEZINA, Tomáš a Jiří DVOŘÁK. *Algoritmy umělé inteligence: Lekce 5*. 2013b.

CHUNG, Shu-Yun a Han-Pang HUANG. Robot Motion Planning in Dynamic Uncertain Environments. *Advanced Robotics* [online]. 2011), č. 25, 849–870 [cit. 2015-02-21]. Dostupné z: http://web.a.ebscohost.com.ezproxy.lib.vutbr.cz/ehost/pdfviewer/pdfviewer? sid=fbf69d8e-8225-4aa6-8b2f-1308dd36b6b4%40sessionmgr4005&vid=1&hid=4207

Exploring the Singleton Design Pattern. *MSDN* [online]. 2002 [cit. 2015-05-20]. Dostupné z: https://msdn.microsoft.com/en-us/library/ee817670.aspx

FERGUSON, Dave, Anthony. STENTZ a Sebastian THRUN. PAO for planning with hidden state. *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004* [online]. IEEE, 2004, 2840-2847 Vol.3 [cit. 2015-01-31]. DOI: 10.1109/ROBOT.2004.1307491. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1307491

FERGUSON, Dave a Anthony STENTZ. *Planning with map uncertainty* [online]. Carnegie Mellon University, 2004b [cit. 2015-04-07].

HANSEN, Eric A. a Rong ZHOU. 2007. Anytime Heuristic Search. *Journal of Artificial Intelligence* [online]. (28) [cit. 2015-05-10]. Dostupné z: https://www.jair.org/media/2096/live-2096-3136-jair.pdf

HANSEN, Eric A. a Shlomo ZILBERSTEIN. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* [online]. 2001, č. 129, s. 35-62 [cit. 2015-01-31]. Dostupné z: http://rbr.cs.umass.edu/papers/HZaij01b.pdf

ITNETWORK. *Generování náhodného bludiště* [online]. 2015 [cit. 2015-05-20]. Dostupné z: http://www.itnetwork.cz/algoritmus-tvorba-nahodneho-bludiste

KAELBLING, Leslie Pack, Michael L. LITTMAN a Anthony R. CASSANDRA. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* [online]. 1998, č. 101, s. 99-134 [cit. 2015-02-28]. Dostupné z: http://www.sciencedirect.com.ezproxy.lib.vutbr.cz/science/article/pii/S000437029800023X#

KOENIG, Sven a Maxim LIKHACHEV. Fast Replanning for Navigation in Unknown Terrain. *354 IEEE TRANSACTIONS ON ROBOTICS* [online]. 2005, (21): 354-363 [cit. 2015-05-24]. Dostupné z: http://ieeexplore.ieee.org.ezproxy.lib.vutbr.cz/stamp/stamp.jsp?tp=&arnumber=1435479

LAVALLE, Steven M. *Planning algorithms* [online]. 2006 [cit. 2015-02-24]. Dostupné z: http://planning.cs.uiuc.edu/

MARTELLI, Alberto a Ugo MONTANARI. Optimizing decision trees through heuristically guided search. *Communications of the ACM* [online]. 1978, vol. 21, issue 12, s. 1025-1039 [cit. 2015-02-12]. DOI: 10.1145/359657.359664.

MICROSOFT CORPORATION. 2015. *Microsoft DreamSpark - Product Visual Studio Professional 2013 with Update 4* [online]. [cit. 2015-05-13]. Dostupné z: https://www.dreamspark.com/Product/Product.aspx?productid=93

MOORE, Andrew a Christopher ATKESON. Prioritized Sweeping: Reinforcement Learning With Less Data and Less Time. *Machine Learning* [online]. 199, s. 103-130 [cit. 2015-04-08]. Dostupné z: http://link.springer.com/article/10.1007%2FBF00993104

MSDN. *Windows API Index (Windows)* [online]. 2015 [cit. 2015-05-19]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/ff818516%28v=vs.85%29.aspx

NILSSON, N.J. *Problem Solving Methods in Artificial Intelligence*. New Yourk: McGraw-Hill, 1971.

PAPADIMITRIOU, Christos H. a Mihalis YANNAKAKIS. Shortest paths without a map. *Theoretical Computer Science* [online]. 1991, č. 84, s. 127-150 [cit. 2015-02-21]. DOI: doi:10.1016/0304-3975(91)90263-2. Dostupné z: http://www.sciencedirect.com.ezproxy.lib.vutbr.cz/science/article/pii/0304397591902632

SHANI, Guy, Joelle PINEAU a Robert KAPLOW. A survey of point-based POMDP solvers. *Auton Agent Multi-Agent Syst* [online]. 2013, s. 1-51 [cit. 2015-04-08]. DOI: 10.1007/s10458-012-9200-2.

STANFORD THEORY. *Heuristics* [online]. 2015 [cit. 2015-05-21]. Dostupné z: http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html

SUTTON, Richard S. a Andrew G. BARTO. 1998. *Reinforcement learning*. London: London : MIT Press. ISBN 0-262-19398-1.

ŠEDA, Miloš. *Graph theory* [online]. 2006 [cit. 2015-02-09]. Dostupné z: http://uai.fme.vutbr.cz/seda/teorie-grafu/TG06_eng.pdf

THE MATHWORKS, INC. 2015. *MATLAB - The Language of Technical Computing* [online]. [cit. 2015-05-13]. Dostupné z: http://www.mathworks.com/products/matlab/index-b.html

THE MATHWORKS. 2015b. *Products and Services - MATLAB and Simulink* [online]. [cit. 2015-05-13]. Dostupné z: http://www.mathworks.com/products/

Umělá inteligence - prohledávání grafu. POPELKA, Ondřej. *Akela.mendelu.cz: studentský server* [online]. 2015 [cit. 2015-03-03]. Dostupné z: https://akela.mendelu.cz/~xpopelka/cs/ui/graf/

VEERA RAGHAVAVAIAH, Gurram. Artificial Intelligence: Problem Reduction with AO* Algorithm. In: *Artificial Intelligence* [online]. 2010 [cit. 2015-04-06]. Dostupné z: http://artificialintelligence-notes.blogspot.cz/2010/07/problem-reduction-with-ao-algorithm.html

Win32++. *Win32++* [online]. 2015 [cit. 2015-05-19]. Dostupné z: http://win32-framework.sourceforge.net/

Win32++. *Win32++* [online]. 2015b [cit. 2015-05-20]. Dostupné z: Dostupné z: http://sourceforge.net/projects/win32-framework/files/Win32%2B%2B/Version%207.8/

ZILBERSTEIN, Shlomo. 1996. Using Anytime Algorithms in Intelligent Systems. *AI Magazine Volume* [online]. (17): 73-83 [cit. 2015-05-09]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.175.8876&rep=rep1&type=pdf