VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

PROHLEDÁVÁNÍ METRICKÉHO PROSTORU S PŘEKÁŽKAMI

SEARCH OF METRIC STATE SPACE WITH OBSTACLES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAKUB LUKÁČ
AUTHOR

VEDOUCÍ PRÁCE — Ing. MARTIN ŠŮSTEK
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Lukáč Jakub**
Obor: Informační technologie
Téma: **Prohledávání metrického prostoru s překážkami**
Search of Metric State Space with Obstacles
Kategorie: Umělá inteligence

Pokyny:

1. Prostudujte metody prohledávání stavového prostoru diskrétního systému.
2. Zvolte alespoň dva různé přístupy pro efektivní řešení úlohy prohledávání stavového prostoru s překážkami.
3. Navrhněte metodu zaměřenou na úlohy v metrickém prostoru, resp. modifikujte na tyto úlohy některý z vybraných přístupů.
4. Navrhněte aplikaci s jednoduchým grafickým uživatelským rozhraním pro prohledávání metrického prostoru s překážkami.
5. Implementujte navrženou aplikaci.
6. Proveďte potřebné experimenty pro porovnání rychlosti a paměťové zátěže obecných přístupů a přístupů založených na metrickém prostoru.

Literatura:

- P. Zezula, Similarity search: the metric space approach. New York: Springer Science Business Media, 2006.
- W. Zhang, State-space search: algorithms, complexity, extensions, and Applications. New York: Springer, 1999.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Šůstek Martin, Ing.**, UITS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Táto práca sa zameriava na prehľadávanie metrického priestoru s prekážkami. Práca vyberie štyri metódy založené na prehľadávaní stavového priestoru a predstaví dva nové algoritmy, ktoré sa pokúsia brať do úvahy prekážky v priestore. Vybrané algoritmy a novo navrhnuté algoritmy sú implementované ako aplikácia v programovacom jazyku Java, aplikácia je priložená v prílohe. Práca predkladá experimenty na priestoroch s rôznymi typmi prekážok pre porovnanie jednotlivých metód.

Abstract

This thesis is focused on search of metric state space with obstacles. The thesis selects four methods based on state space search and presents two new algorithms, which will try to consider an obstacles in metric space. The selected methods and the new designed algorithms are implemented as an application in programming language Java, application is also part of the thesis. The thesis presents experiments on metric spaces with various kind of obstacles for comparison of individual methods.

Kľúčové slová

Prehľadávanie stavového priestoru, Prehľadávanie A*, Metrický priestor, Prehľadávanie metrického priestoru, Metrika, Java

Keywords

State space search, A* search, Metric Space, Search of Metric State Space, Metric, Java

Citácia

LUKÁČ, Jakub. *Prohledávání metrického prostoru s překážkami.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Šůstek

Prohledávání metrického prostoru s překážkami

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Martina Šůsteka. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

. . . . . .
Jakub Lukáč
16. mája 2018

Poďakovanie

Ďakujem pánovi Ing. Martinovi Šůstekovi za cenné rady, ktoré mi poskytol na konzultáciach pri riešení tejto bakalárskej práce.

Obsah

1 Úvod **3**

2 Prehľadávanie stavového priestoru **4**
2.1 Hodnotiace kritéria metód 4
2.2 Neinformované metódy 5
2.2.1 Breadth-first search 5
2.2.2 Depth-first search 5
2.2.3 Depth-limited search 6
2.2.4 Iterative deepening search 6
2.2.5 Bidirectional search 6
2.3 Informované metódy 7
2.3.1 Best-first search 7
2.3.2 Metódy lokálneho prehľadávania 8

3 Metrický priestor **9**
3.1 Pojem metrického priestoru 9
3.2 Prehľadávanie metrického priestoru 10

4 Zvolené metódy prehľadávania metrického priestoru **11**
4.1 A* search 11
4.2 Greedy search 12
4.3 Weighted A* 13
4.4 Beam search 13
4.5 Zjednodušenie priestoru 14
4.6 Obojsmerné A* so zohľadnením prekážok 15

5 Implementácia **18**
5.1 Implementácia algoritmov 18
5.2 Užívateľské rozhranie aplikácie 19
5.3 Ovládanie 19

6 Experimenty **21**
6.1 Experiment 1 21
6.2 Experiment 2 22
6.3 Experiment 3 23
6.4 Experiment 4 24
6.5 Experiment 5 25
6.6 Zhodnotenie výsledkov 26

7 Záver **28**

Literatúra **30**

A Obsah priloženého DVD **31**

Kapitola 1

Úvod

V dnešnej dobe je nájdenie cesty medzi dvomi miestami pomerne častou úlohou – či už to platí pre ľudí, ktorý sa chcú dostať z jedného miesta do druhého čo najrýchlejšie a najefektívnejšie alebo pre umelú inteligenciu v počítačovej hre, ktorá sa snaží prekonať prekážky a dostať k hráčovi. V priebehu času vzniklo veľké množstvo algoritmov, ktoré sa túto úlohu snažia riešiť a ktoré sa od seba môžu veľmi líšiť hlavne poskynutými výsledkami.

Cieľom tejto práce je predovšetkým navrhnúť metódu pre prehľadávanie metrického priestoru s prekážkami, navrhnutú metódu implementovať do aplikácie, ktorá bude prehľadávanie priestoru zobrazovať vhodnou formou a navrhnutú metódu otestovať a porovnať s vybranými efektívnymi metódami založenými na prehľadávaní stavového priestoru.

Kapitola 2 uvádza do problematiky prehľadávania stavového priestoru a poskytuje prehľad základných neinformovaných a informovaných algoritmov. V kapitole 3 je vysvetlený pojem metrického priestoru. Kapitola 4 poskytuje detailnejší pohľad na vybrané metódy pre prehľadávanie metrického priestoru. Obsahuje taktiež popis dvoch navrhnutých metód, ktoré je možné použiť na prehľadávanie. Kapitola 5 uvádza stručný prehľad o aplikácii implementovanej v jazyku Java. Experimenty pre porovnanie jednotlivých metód sú uvedené v kapitole 6, kde sú na množinách bodov s prekážkami prevedené prehľadávania priestoru s rôznymi kombináciami metrík a heuristických funkcii. Časť 6.6 potom zhŕňa výsledky prevedených experimentov.

V závere práce je zhodnotená celá práca a sú tam navrhnuté ďalšie úpravy implementovanej aplikácie a možné vylepšenia, o ktorých by sme v budúcnosti mohli uvažovať.

Kapitola 2

Prehľadávanie stavového priestoru

Algoritmy založené na prehľadávaní stavového priestoru sú jedným zo základných typov metód pre riešenie úloh v oblasti umelej inteligencie. Stavový priestor [5] môžeme definovať ako dvojicu (S, O), kde S je množina všetkých stavov úlohy a O je množina všetkých operátorov, ktorými je možné stavy úlohy meniť. Úloha je potom definovaná ako dvojica (s_0, G), kde $s_0 \in S$ značí počiatočný stav a $G \subset S$ je množina všetkých cieľových stavov danej úlohy. Riešením úlohy je potom postupnosť operátorov, ktoré postupným aplikovaním na stav s_0 dostanú úlohu do jedného zo stavov z množiny G.

Podľa toho, akým spôsobom prebieha prehľadávanie stavového priestoru sa metódy rozdeľujú na dve základné skupiny: neinformované, ktoré pri prehľadávaní nevyužívajú žiadne informácie o úlohe a informované, ktoré majú za cieľ prehľadávanie čo najviac zrýchliť pomocou akýchkoľvek informácií o cieľovom stave úlohy. Nasledujúce strany vychádzajú z [3].

2.1 Hodnotiace kritéria metód

Keďže existuje veľké množstvo metód, ktoré riešia úlohy prehľadávaním stavového priestoru, bolo nutné zaviesť aspoň základné kritéria, ktorými budú jednotlivé metódy hodnotené a porovnávané. Hoci niektoré z nižšie rozoberaných metód sa líšia na prvý pohľad len veľmi málo, často sa aj malá zmena algoritmu prejaví v niektorom z nasledujúcich kritérií:

- **Časová náročnosť** – časová náročnosť bude vyjadrená pomocou $O(f)$, čo značí maximálnu časovú náročnosť metódy a f je funkcia reprezentujúca závislosť časových prostriedkov pre vyriešenie úlohy na niektorých základných parametrov úloh. Týmito parametrami sú napríklad b – priemerný počet bezprostredných nasledovníkov každého stavu, d – hĺbka najlepšieho riešenia, teda najmenší počet stavov, ktoré je nutné prejsť pri ceste z počiatku do cieľa.

- **Pamäťová náročnosť** – priestorová náročnosť bude opäť vyjadrená pomocou $O(f)$, kde f je závislosť priestorových nárokov algoritmu na najhoršie možné parametre úlohy. Pamäťovú náročnosť môžeme chápať ako maximálny počet stavov, ktoré môžu byť počas prehľadávania súčasne uložené v pamäti.

- **Úplnosť** – úplná metóda nájde riešenie vždy, ak existuje. Hoci sa môže zdať požiadavok na úplnosť metódy ako absolútne kritický, nie je tomu vždy tak. Existujú rôzne typy úloh, pri ktorých sme ochotní sa zmieriť aj s tým, ak metóda riešenie nenájde. Použitie niektorej z úplných metód by mohlo byť v takých prípadoch kontraproduktívne vďaka nepriaznivej časovej alebo pamäťovej náročnosti.

- **Optimálnosť** – ak je metóda optimálna, znamená to, že nájdené riešenie je vždy najlepšie zo všetkých. Optimálnosť si môžeme predstaviť napríklad ako najkratšiu cestu na mape medzi dvomi mestami. Neoptimálnosť by v takom prípade mohla znamenať cesta z Brna do Bratislavy cez Varšavu.

2.2 Neinformované metódy

Neinformované alebo niekedy označované ako slepé algoritmy (citácia IZU opory) je skupina metód, ktoré pre riešenie úloh nepoužívajú žiadne informácie o cieľovom stave ani žiadne prostriedky, ako stavy ohodnotiť. Prehľadávanie neinformovanými metódami teda znamená systematické prechádzanie stavového priestoru až kým nenájdu riešenie.

Rozumné využitie neinformovaných metód prichádza do úvahy hlavne pre menšie stavové priestory alebo v prípade, že nie je možné použiť vhodnú heuristickú funkciu k informovaným metódam.

2.2.1 Breadth-first search

Jedná sa o základnú neinformovanú metódu, ktorá patrí medzi najjednoduchšie. Slovne by sa dal algoritmus popísať nasledujúcim spôsobom:

1. Zostrojte frontu OPEN a umiestnite do nej počiatočný stav.
2. Ak je fronta OPEN prázdna, úloha nemá riešenie a ukončite toto prehľadávanie ako neúspešné.
3. Vyberte z čela fronty OPEN prvý stav.
4. Ak je vybraný stav cieľovým, ukončite prehľadávanie ako úspešné a vráťte cestu od koreňového stavu k cieľovému (vracia sa postupnosť stavov a operátorov). Inak pokračujte.
5. Vybraný stav expandujte, všetkých jeho bezprostredných následníkov umiestnite do fronty OPEN a vráťte sa na bod 2.

Algoritmus teda slepo prechádza celý strom stavového priestoru, pričom vždy prejde celú jednu úroveň stromu a až po jej prejdení sa posunie o úroveň nižšie. Týmto spôsobom systematicky prechádza celý priestor. V prípade, že je počet bezprostredných následníkov konečný, potom je algoritmus BFS **úplný** a **optimálny**.

Aj keď sa môže zdať, že metóda poskytuje kvalitné výsledky, jej použitie výrazne obmedzuje časová zložitosť $O(b^{d+1})$, ktorá je exponenciálna. Keďže metóda musí uchovávať v pamäti všetky stavy, ktoré doteraz rozgenerovala, pretože nevie, ktoré z nich na konci budú potrebné pre získanie riešenia, je jej priestorová zložitosť rovná časovej – $O(b^{d+1})$.

Exponenciálna časová a priestorová zložitosť robí tento jednoduchý algoritmus prakticky nepoužiteľný pre riešenie zložitejších úloh. Už v prípade pre $b = 7$ a $d = 5$ vychádza najhorší počet rozgenerovaných stavov na 16807, ak by sme prekročili v oboch prípadoch hodnotu 10, už najhorší počet stavov narastá na desiatky miliárd.

2.2.2 Depth-first search

Túto metódu dostaneme upravením algoritmu BFS, ak ako open namiesto fronty použijeme zásobník. Pri použití zásobníku metóda postupuje vždy do čo najväčšej hĺbky čo najďalej

od počiatočného stavu. Až v prípade dosiahnutia konca stromu sa vracia a prehľadáva podobným spôsobom ďalšie stavy.

Vďaka tomu, že pri prehľadávaní do hĺbky môže dôjsť k zacykleniu, teda neustáleho generovania tých istých stavov stále dookola je časová náročnosť metódy ešte horšia, ako BFS. Časové nároky sú dané vzorcom $O(b^m)$, kde m je maximálna prehľadávaná hĺbka stromu, ktorá teoreticky môže narásť až do nekonečna. Navyše kvôli riziku možného zacyklenia nie je metóda ani úplná, ani optimálna.

Naproti tomu, priestorová náročnosť metódy DFS je lineárna ako $O(bm)$, ím ju v tomto ohľade činí oveľa priaznivejšou, ako BFS. Napriek zjavnej nevýhode, ktorou je neúplnosť a neoptimálnosť stále existujú prípady, kedy je výhodné použiť DFS. Môže k tomu viesť napríklad úloha, kde je použitie BFS kvôli nepriaznivým priestorovým vlastnostiam nereálne. Prípadne je možné algoritmus DFS upraviť takým spôsobom, aby sa zabránilo zacykleniu, potom však môže stratiť niektoré zo svojich zjavných výhod.

2.2.3 Depth-limited search

Tento algoritmus môžeme použiť vtedy, keď dokážeme odhadnúť, v akej hĺbke sa nachádza riešenie. Ak pri metóde DFS obmedzíme maximálnu hĺbku zanorenia na l, dostaneme metódu DLS. Táto drobná úprava dokáže odstrániť problém s tým, ako sa pri DFS môžu generovať stále dookola tie isté stavy.

Algoritmus podobne ako DFS nie je optimálny – nájdené riešenie sa môže nachádzať príliš hlboko, zatiaľ čo lepšie (optimálne) nebolo nájdené, pretože sa nachádza príliš blízko koreňa a neprišiel naňho rad. Pretože riešenie sa môže nachádzať aj vo väčšej hĺbke, ktorú určuje parameter l a DLS sa k nemu nemusí dostať, metóda DLS nie je ani úplná.

Časová náročnosť algoritmu zostáva exponenciálna, daná výrazom $O(b^l)$, pamäťová náročnosť je podobne ako pri DFS lineárna, $O(bl)$. Vďaka pomerne priaznivej pamäťovej náročnosti sa ani neúplnosť algoritmu nemusí zdať ako veľmi zlá vlastnosť. Existujú úlohy, kde je možné pomerne presne predpovedať, v akej hĺbke sa môže riešenie nachádzať.

2.2.4 Iterative deepening search

V prípade, že nie je možné presne stanoviť hranicu, v akej hĺbke sa nachádza riešenie a pre pamäťové nároky nie je možné použiť algoritmus BFS, je možné tento problém vyriešiť algoritmom IDS. Princíp metódy spočíva v opakovanom spúšťaní algoritmu DLS s limitom, ktorý sa pri každom spustení zvýši.

Tým, že nie je potrebné odhadovať parameter l ako v metóde DLS, ale algoritmus sám si túto hodnotu navyšuje je zrejmé, že metóda IDS je úplná a optimálna. Hoci sa na prvý pohľad môže zdať opakované prehľadávanie ako mimoriadne neefektívne, časová náročnosť metódy je veľmi podobná BFS – $O(b^d)$. Pretože pamäťová náročnosť algoritmu je lineárna, $O(bd)$, je použitie metódy IDS ešte výhodnejšie ako DFS.

Vďaka vyššie uvedeným vlastnostiam môžeme konštatovať, že použitie metódy IDS je skutočne výhodné a to napriek tomu, že na prvý pohľad je opakované prehľadávanie do stále väčšej hĺbky nezmyselné.

2.2.5 Bidirectional search

Algoritmus bidirectional search, preložené ako obojsmerné prehľadávanie je metóda založená na rozbehnutie prehľadávania od poiatku k cieľu a od cieľu k počiatku. Tieto prehľadávania

prebiehajú paralelne – ak z jednej strany nájdený stav, ktorý už bol preskúmaný druhou stranou, dôjde k spojeniu cesty a nájdeniu riešenia.

Ak v obidvoch smeroch používame na prehľadávanie algoritmus BFS a je v každom smere preskúmaná len polovičná hĺbka, potom bude časová zložitosť generovania stavov rovná $O(b^{d/2})$. Negatívny vplyv na časové vlastnosti algoritmu môže mať neustála kontrola, či sa skúmaný stav už nepreskúmal druhou stranou.

Použitie obojsmerného prehľadávania je možné pri splnení dvoch základných požiadavkoch. Je potrebné presne poznať cieľový stav, aby z neho bolo možné začať prehľadávať smerom k cieľu. Ak máme len nejasné predstavy o cieľovom stave, nie je možné obojsmerné prehľadávanie použiť. Opačným príkladom môže byť, ak máme viac cieľových stavov, v tom prípade je nutné sa rozhodnúť, ktorý z nich pre prehľadávanie použiť. Ďalšou podmienkou je, aby operátory stavového priestoru boli **reverzibilné**, takže aby metóda mohla okrem generovania bezprostredných následníkov jednoducho generovať aj bezprostredných predchodcov.

2.3 Informované metódy

Zásadným rozdielom informovaných metód oproti neinformovaným je v tom, že informované pri riešení úloh používajú nejaké informácie o cieľovom stave a prostriedky k tomu, ako ohodnotiť aktuálny stav. Ohodnocovanie stavov prebieha na základe ohodnocovacej funkcie, ktorej hlavnou zložkou je takzvaná **heuristická funkcia**. Typickou vlastnosťou heuristickej funkcie je, že sa jej hodnota zmenšuje, čím bližšie k cieľu je aktuálne hodnotený stav. Nulová je práve v cieľovom stave. Čím lepšie výsledky poskytuje dodaná heuristická funkcia, tým rýchlejšie môže daný algoritmus nájsť kvalitné riešenie. Naopak pri heuristickej funkcii, ktorá poskytuje nezmyselné hodnoty (napríklad konštantu) to algoritmus môže zhoršiť, až sa z neho stane niektorá z neinformovaných metód.

Informované metódy sa v základe delia na metódy typu best-first search a metódy lokálneho prehľadávania.

2.3.1 Best-first search

Názov best-first search označuje niekoľko algoritmov založených na výbere najlepšie ohodnoteného stavu. Kľúčovým pojmom je ohodnocovacia funkcia, na základe ktorej sa vždy vyberie stav s najmenším ohodnotením. Podľa toho, aký má táto funkcia tvar rozlišujeme jednotlivé metódy založené na best-first search. Nasleduje popis algoritmu:

1. Zostrojte zoznam OPEN a vložte doňho počiatočný stav vrátane jeho ohodnotenia.
2. Ak je zoznam OPEN prázdny, ukončite prehľadávanie ako neúspešné. Inak pokračujte.
3. Vyberte zo zoznamu OPEN stav s najlepším (najnižším) ohodnotením.
4. Ak je vybraný stav cieľovým, ukončite prehľadávanie ako úspešné a vráťte cestu od koreňového stavu k cieľovému (vracia sa postupnosť stavov a operátorov). Inak pokračujte.
5. Vybraný stav expandujte, všetkých jeho bezprostredných následníkov, ktorí nie sú jeho predkami umiestnite do zoznamu OPEN a to vrátane ich ohodnotenia. V prípade, že sa niektorý stav nachádza v OPEN viackrát, ponechajte iba stav s najlepším ohodnotením, ostatné odstráňte. Vráťte sa na bod 2.

Zásadným rozdielom oproti neinformovaným metódam je ohodnocovanie stavov na základe funkcie, ktorá má tvar:

$$f(n) = g(n) + h(n) \tag{2.1}$$

kde $g(n)$ je skutočná prejdená cesta od počiatku k aktuálnemu stavu a $h(n)$ je heuristická funkcia, ktorá *odhaduje* ostávajúcu vzdialenosť k cieľu. V prípade, že ohodnocovacia funkcia používa obe tieto zložky, tak hovoríme o metóde A*. Ak zanedbáme zložku $g(n)$ a algoritmus sa tak bude riadiť v prehľadávaní len heuristikou, dostaneme metódu greedy search. V prípade, že naopak heuristická funkcia bude poskytovať nulový výsledok dostaneme algoritmus UCS, ktorý sa tým pádom radí medzi neinformované algoritmy, hoci patrí medzi metódy typu best-first search.

2.3.2 Metódy lokálneho prehľadávania

Metódy lokálneho prehľadávania sú algoritmy, ktoré sú použité pre úlohy, ktorých riešením je iba nájdenie cieľového stavu a vlastná cesta je pritom bezvýznamná. Na rozdiel od vyššie popísaných metód neprehľadávajú stavový priestor systematicky, snažia sa len nájsť optimálny stav.

Napriek tomu, že neprehľadávajú celý priestor majú metódy lokálneho prehľadávania dve veľké výhody: majú celkom zanedbateľnú pamäťovú náročnosť a často dospejú k prijateľnému riešeniu v rozsiahlych stavových priestoroch aj vtedy, keď je použitie vyššie vymenovaných algoritmov nemožné. Ako príklad metód lokálneho prehľadávania môžeme uviesť dve metódy, ktoré sa líšia hlavne spôsobom výberu ďalšieho stavu:

- **Hill climbing** – metóda voľne preložená ako lezenie do kopca používa pre ohodnotenie aktuálneho stavu iba heuristickú funkciu $h(n)$. Obykle však neohodnocuje vzdialenosť do cieľa, ale kvalitu daného stavu. Potom čím kvalitnejší je daný stav, tým vyššie hodnotenie mu bude priradené. Algoritmus pre expanziu vyberie stav s najlepším ohodnotením. Výsledkom prehľadávania však nemusí byť celkové globálne riešenie, pretože môže uviaznuť v lokálnom extréme.

- **Metóda simulovaného žíhania** – metóda bola navrhnutá ako pokus vyriešiť problém algoritmu hill climbing, ktorý môže uviaznuť v lokálnych extrémoch. Aby sa uviaznutiu v extréme vyhla, dovoľuje v určitých prípadoch vybrať za ďalší aj stav, ktorý sa v danom okamihu nezdá ako najlepší. Heuristika berie do úvahy aj takzvanú teplotu, ktorá sa pri hľadaní postupne znižuje podľa predom stanovenej tabuľky. Pri vysokých teplotách je pravdepodobnosť výberu menej kvalitného nasledovníka vysoká, naopak pri tepote blízko nuly sa jedná prakticky o algoritmus hill climbing.

Kapitola 3

Metrický priestor

Táto kapitola je venovaná popisu metrického priestoru a odpovedajúcim pojmom. Metrický priestor je matematická štruktúra, pomocou ktorej je možné zobecniť a definovať pojem vzdialenosti [1].

3.1 Pojem metrického priestoru

Metrický priestor [4] môžeme definovať ako dvojicu $M = (D, d)$, kde M je ľubovoľná neprázdna množina a d je takzvaná metrická funkcia. Metrická funkcia, skrátene metrika je zobrazenie $d : D \times D \mapsto \mathbb{R}$, kde musí platiť:

$\forall x, y \in D, d(x, y) > 0$	nezápornosť
$\forall x, y \in D, d(x, y) = d(y, x)$	symetria
$\forall x, y \in D, x = y \leftrightarrow d(x, y) = 0$	totožnosť
$\forall x, y, z \in D, d(x, z) \leq d(x, y) + d(x, z)$	trojuholníková nerovnosť

Pre lepšiu definíciu metrického priestoru je zvykom definovať ekvivalentné axiómy:

$\forall x, y \in D, d(x, y) > 0$	nezápornosť
$\forall x, y \in D, d(x, y) = d(y, x)$	symetria
$\forall x \in D, d(x, x) = 0$	reflexivita
$\forall x, y \in D, x \neq y \rightarrow d(x, y) > 0$	pozitívnosť
$\forall x, y, z \in D, d(x, z) \leq d(x, y) + d(x, z)$	trojuholníková nerovnosť

V prípade, že metrika nespĺňa axióm pozitívnosti, hovoríme o takzvanej pseudometrike.

Ako bolo vysvetlené vyššie, metrika je zobrazenie, ktoré ľubovoľným dvom bodom z D priradí číselnú hodnotu, ktorú môžeme zjednodušene nazvať ako vzdialenosť. V priestore $\mathbb{R}^n$ je možné na množinu bodov D zaviesť veľké množstvo metrík splňujúce dané axiómy. Nižšie nasledujú definície troch metrík (súhrnne niekedy označované ako Minkowského vzdialenostné funkcie), ktoré budú použité v nasledujúcich kapitolách pre prehľadávanie metrického priestoru:

- **Manhattanská metrika** – definovaná podľa vzorca:

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

- **Euklidovská metrika** – definovaná podľa vzorca:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}$$

- **Čebyševská metrika** – označovaná ako maximálna alebo šachovnicová metrika definovaná podľa vzorca:

$$d(x, y) = max\{|x_1 - y_1|, |x_2 - y_2|, \ldots |x_n - y_n|\}$$

3.2 Prehľadávanie metrického priestoru

Pre účely tejto práce môžeme prehľadávanie stavového priestoru definovať ako úlohu, kde je v metrickom priestore M zadaný počiatočný bod, cieľový bod a riešením je postupnosť bodov, ktoré vedú z počiatku do cieľa. V tomto prípade môže množina priestor M obsahovať takzvané prekážky – teda body, ktorými výsledná cesta nemôže viesť a ktoré je potrebné obchádzať.

Kapitola 4

Zvolené metódy prehľadávania metrického priestoru

Algoritmy na prehľadávanie stavového priestoru ponúkajú jasného kandidáta na to, ktorý použiť na prehľadávanie metrického priestoru – A*. Nenašiel som ďalší algoritmus, ktorý by sa zásadne líšil od A* a zároveň by poskytoval rovnako kvalitné výsledky. Preto ostatné vybrané algoritmy majú s A* veľa spoločného, väčšina z nich môže v rýchlosti predstihnúť A*, ich výsledky však nie sú také kvalitné. Pri štúdiu týchto metód potom prišlo na návrh vlastného riešenia tak, aby sa z vybraných metód zobralo to najlepšie. Navrhol som najprv jednu metódu, ktorá je založená na transformácii priestoru do jednoduchšej formy, čím sa môže pri prehľadávaní ušetriť čas. Táto metóda je však použiteľná len teoreticky, preto som navrhol druhú metódu, ktorá kombinuje algoritmy A* a bidirectional search. Druhá metóda (pre účely práce pomenovaná ako BMA*) si medzi sebou predáva informácie o tom, kde sa nachádza nejaká prekážka a pokúsi sa ju prejsť z druhej strany.

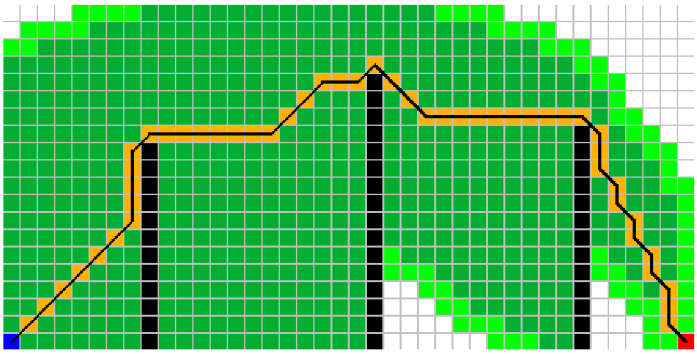
4.1 A* search

Algoritmus A* [3] je najznámejší a najpoužívanejší algoritmus pre riešenia úloh prehľadávaním stavového priestoru. Jedná sa o informovanú metódu typu best-first search, v ktorej má ohodnocovacia funkcia tvar daný vzorcom 2.1, kde heuristická funkcia $h(n)$ je tzv. **spodným odhadom** skutočnej cesty od ohodnocovaného stavu k cieľu. To znamená, že odhad musí byť menší alebo rovný skutočnej ceste. V prípade, že je splnená táto podmienka, hovoríme o prípustnej heuristickej funkcii. A* je úplná a optimálna metóda, výrazne však závisí od použitej heuristiky – ak je napríklad nulová (čo stále spĺňa požiadavok o spodnom odhade), vlastnosti A* sú degradované na neinformovanú metódu. Naopak, pri použití dobrej heuristiky s dobrým skutočným spodným odhadom sú expandované len stavy okolo optimálnej cesty.

Príklad nájdenej cesty pomocou algoritmu A* je možné vidieť na obrázku 4.1. Modrý bod značí štart, červený bod cieľ a čiernou farbou sú vyznačené prekážky. Výsledok prehľadávania dopĺňajú body svetlozelenej farby, ktoré patria do fronty open a body tmavozelenej farby značia už navštívené body. Metrický priestor použití pre tento príklad používal euklidovskú metriku a pre prehľadávanie ako heuristickú funkciu euklidovskú vzdialenosť[1]. Na prvý pohľad je vidieť, že nájdená cesta je optimálna. Keďže sa však medzi štartom a cieľom nachádzali tri prekážky, bolo nutné prehľadať väčšie množstvo bodov. V prípade, že by

[1] Všetky ďalšie príklady použitia algoritmov v tejto kapitole používajú rovnakú konfiguráciu.

v danom príklade nebola žiadna prekážka, výsledkom by bola rovná čiara od počiatku ku cieľu bez žiadnych zbytočne prehľadaných bodov.



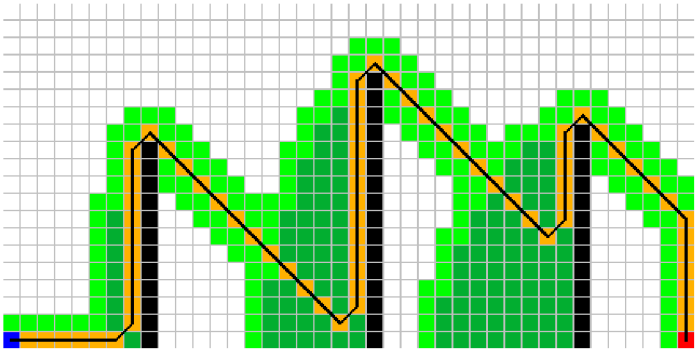
Obr. 4.1: Príklad optimálnej nájdenej cesty algoritmom A* search

4.2 Greedy search

Algoritmus greedy search [3] je informovaný algoritmus typu best-first search, ktorý používa ohodnocovaciu funkciu

$$f(n) = h(n) \tag{4.1}$$

Pri expanzií stavov sa tedia riadi len heuristickou funkciou a vyberá vždy stav, ktorý má heuristiku najmenšiu. Od tejto vlastnosti pochádza aj meno greedy (*hladný, nenásytný*), pretože pri prehľadávaní sa „nenásytne“ uberá za svojim cieľom. Z dôvodu zanedbania užitočného údaju o prejdenej ceste – funkcii $g(n)$ je algoritmus obecne neoptimálny a v prípade, že nie je ošetrené neustále generovanie rovnakých stavov nie je dokonca ani **úplný**.



Obr. 4.2: Príklad neoptimálnej cesty nájdenej algoritmom Greedy search

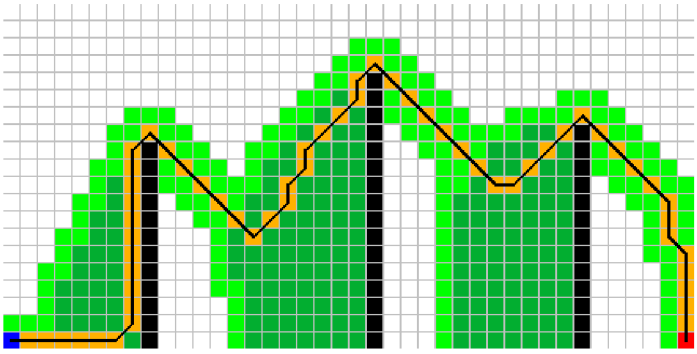
Príklad nájdenej cesty algoritmom greedy search je na obrázku 4.2. Jediným pozitívom je, že pre nájdenie cetsy bolo potrebné prejsť oveľa menšie množstvo bodov, ako pri algoritme A*. Kvalita získanej cesty je však najhoršia zo všetkých príkladov uvedených v tejto kapitole. Na obrázku je jasne vidieť, ako veľmi môže ovplyvniť zanedbanie zložky $g(n)$ v ohodnocovacej funkcii. Hoci aj algoritmus A* prehľadával rovnaké stavy ako greedy, zanedbanie hodnoty prejdenej cesty drasticky ovplyvnilo výslednú cestu.

4.3 Weighted A*

Weighted A* (váhované A*) [2] je variantou algoritmu A*, ktorá používa upravenú ohodnocovaciu funkciu:

$$f'(n) = g(n) + \varepsilon \cdot h(n) \tag{4.2}$$

kde ε predstavuje váhu, ktorou je pri každom ohodnotení stavu prenásobená hodnota heuristiky, čím heuristika v závislosti na hodnote váhy naberá na dôležitosti. Algoritmus vďaka novej ohodnocovacej funkcii stráca optimálnosť, pričom ale oproti A* môže prehľadávanie značne zrýchliť. Na obrázku 4.3 s hodnotou $\varepsilon = 5$ je možné si všimnúť, že kvalita nájdeného riešenia nie je veľká (so zvyšujúcou hodnotou ε má tendenciu čoraz viac pripomínať výsledok algoritmu greedy search), ale množstvo prehľadaných stavov oproti A* pokleslo.

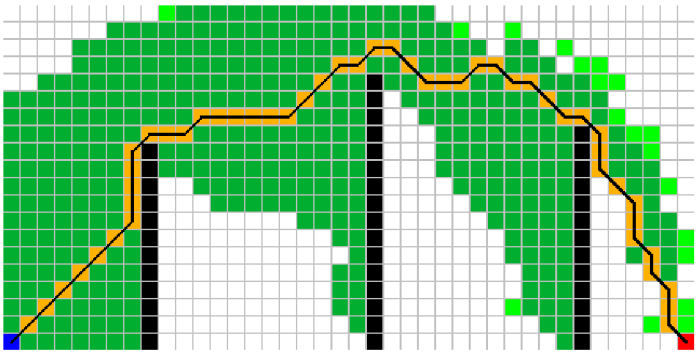


Obr. 4.3: Príklad použitia weighted A* s $\varepsilon = 5$

4.4 Beam search

Názov beam search [2] zastrešuje dve hlavné kategórie algoritmov tohoto typu. Ak hovoríme o **best-first beam search**, prehľadávanie prebieha presne ako v prípade best-first search s tým rozdielom, že ak veľkosť zoznamu alebo fronty open prekročí stanovenú hranicu, budú najmenej kvalitné stavy zahodené. Vrátiť sa do open tieto stavy budú môcť v priebehu prehľadávania až potom, keď sa veľkosť open zníži pod hranicu. Ďalším druhom agoritmu beam search je **breadth-first beam search**. V tomto prípade prebieha prehľadávanie totožne s algoritmom breadth-first search, avšak na každej hlbšej úrovni je expandované len určité množstvo stavov určené konštantou.

V rámci tejto práce budeme pod pojmom beam search označovať best-first beam search ako variantu algoritmu A*. Ako je uvedené vyššie, algoritmus je vždy spustený s parametrom β, ktorý určuje kapacitu fronty open. V prípade, že je β nastavená na nekonečnú hodnotu (alebo bude mať hodnotu $|D|$), prehľadávanie bude úplne zhodné s algoritmom A*. Zahadzovanie menej sľubných stavov potom nastáva, keď veľkosť open prekročí tento parameter, čím sa beam search stáva neoptimálnym algoritmom. Ako je možné vidieť na obrázku 4.4, kde $\beta = 20$, algoritmus za cenu menšej neoptimality našiel cestu o niečo rýchlejšie ako A*.



Obr. 4.4: Príklad použitia beam search s $\beta = 20$

4.5 Zjednodušenie priestoru

Pre účely tohoto algoritmu budeme predpokladať metrický priestor, ktorého body sa nachádzajú na **celočíselných** pozíciach (ako príklad môžeme uviesť $D = \{[0,0],[0,1],\ldots[10,10]\}$). Motiváciou vzniku tejto metódy je to, že aj efektívne algoritmy v priestore bez prekážok musia pri ceste do cieľa prekonať stovky alebo tisíce bodov, hoci v ceste nemusí stáť nijaká prekážka a tento algoritmus sa pokúsi tento problém obísť. Pre vysvetlenie samotného algoritmu je potrebné najskôr definovať niekoľko pojmov, s ktorými sa bude ďalej pracovať:

Definícia 1 *Nech $A, B, A \neq B$ sú body. Ak platí $|A_x - B_x| \leq 1 \wedge |A_y - B_y| \leq 1$, potom o bodoch A, B môžeme povedať, že sú* susediace. *Ak platí $|A_x - B_x| = 1 \wedge |A_y - B_y| = 1$, potom sú body A, B susedné* diagonálne.

Definícia 2 *Nech (D, d) je metrický priestor. Ak existuje bod $A \in D$ a prekážka B, ktoré sú susedné diagonálne a zároveň neexistuje prekážka $C \neq B$ ktorá susedí zároveň s bodmi A, B, potom môžeme povedať, že bod A je* rohový bod.

Definícia 3 Graf rohových bodov *nad metrickým priestorom (D, d) je dvojica (G, V), kde G je množina rohových bodov obsahujúca aj počiatočný a cieľový bod a V je množina prepojov medzi bodmi z G. Prepoj je definovaný ako množina $\{A, B\}$, kde $A, B \in G$, pričom musí pre každý bod C, kde $min(A_x, B_x) \leq C_x \leq max(A_x, B_x) \wedge min(A_y, B_y) \leq C_y \leq max(A_y, B_y)$ platiť, že nesmie byť prekážkou. V potom obsahuje všetky prepoje medzi bodmi z G.*

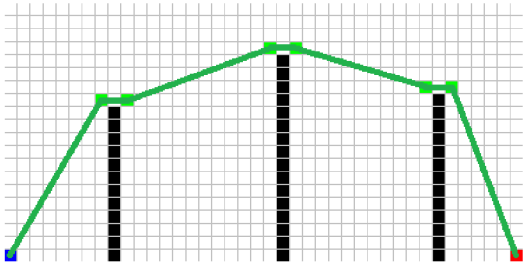
Zjednodušený popis algoritmu:

1. Zostrojte graf rohových bodov – nájdite všetky rohové body priestoru a všetky prepoje medzi nimi.
2. Začnite získaný graf prehľadávať algoritmom A*.
3. Ak bolo prehľadávanie úspešné, získajte z vrátenej cesty cestu skutočnú.

Z uvedeného popisu algoritmu vyplýva niekoľko dôležitých skutočností. Ak budeme uvažovať jednoduchý prípad priestoru, ktorý obsahuje milión bodov, v priestore nie sú rozmiestnené nijaké prekážky, ale cieľový bod je priamo susediaci s počiatočným bodom, algoritmus túto situáciu nezachytí. To znamená, že najskôr bude musieť preskúmať celý priestor, aby zistil, ktoré body sú rohové a podľa toho zostrojil graf rohových bodov. Až pri

prehľadávaní grafu zistí, že existuje len jeden prepoj – medzi počiatočným bodom a cieľom a túto jednoduchú výslednú cestu vráti. Takýto prípad je dokonca jedným z tých najhorších možných spôsobov, ako riešiť konkrétny problém nájdenia cesty v priestore. Priaznivejšie výsledky by mala dokonca akákoľvek slepá metóda prezentovaná v 2.2.

Preto je nutné zdôrazniť, že vyššie uvedený algoritmus je nevhodný aplikovať na priestor, ktorý je pri danej úlohe neznámy, teda riešiteľ nemá žiadne informácie o tom, ako sú v danom priestore rozmiestnené prekážky. Ak však budeme uvažovať priestor, v ktorom poznáme rozmiestnenie prekážok (čo však nemusí byť častý prípad), je možné si z rozmiestnenia prekážok odvodiť polohu všetkých rohových bodov. Keďže prepoj je definovaný len tým, aby sa medzi bodmi nenachádzala nijaká prekážka, je aj kompletné zostrojenie grafu zvládnuteľnou úlohou.



Obr. 4.5: Graf rohových bodov

Príklad grafu rohových bodov je možné vidieť na obrázku 4.5, kde sa graf skladá len zo siedmich prepojov.

4.6 Obojsmerné A* so zohľadnením prekážok

Algoritmy založené na best-first search ponúkajú pri narazení na prekážku len jedinú reakciu – jej obídenie. V takomto prípade často narastá počet prehľadaných bodov a nepríjemným javom niektorých neoptimálnych metód je konvergencia do lokálneho optima, čím môže vzniknúť veľmi neoptimálne riešenie (4.2). Algoritmus A* si síce aj pri konfrontácií s prekážkami zachová svoju optimálnosť, ale sprievodným javom je zvyšujúci sa počet prehľadaných bodov.

Riešením tejto situácie môže byť spojenie algoritmov A* a bidirectional search. Avšak kým obyčajný obojsmerný prístup nie je samostatná metóda, ale skôr paralelné rozbehnutie prehľadávania od štartu a od cieľa, v tejto metóde nastane dôležitá zmena. Modifikovaná kombinácia týchto dvoch algoritmov (pre účely tejto práce sa na ňu budem odkazovať ako BMA*, čo je skratka z Bidirectional Metric A*) používa upravené ohodnocovacie funkcie – v závislosti na tom, z ktorej strany práve prehľadávanie prebieha. Pri prehľadávaní smerom od štartovacieho bodu má ohodnocovacia funkcia tvar

$$f'(n) = g(n) + h_{tmpGoal}(n) \tag{4.3}$$

kde $g(n)$ je prejdená vzdialenosť bodu n od počiatočného bodu a $h_{tmpGoal}(n)$ je upravená heuristická funkcia. Na rozdiel od ostatných informovaných metód nie je počítaná vždy ku jednému pevnému bodu (typicky cieľový bod), ale závisí od aktuálneho nastavenia pomocného bodu *tmpGoal*, ktorý je definovaný pri prehľadávaní. Analogicky môžeme definovať aj druhú ohodnocovaciu funkciu, ktorá je použitá pri prehľadávaní smerom od cieľového bodu:

$$f'(n) = g(n) + h_{tmpStart}(n) \tag{4.4}$$

kde $h_{tmpStart}(n)$ je upravená heuristická funkcia, ktorá odhaduje vzdialenosť bodu n od pomocného bodu *tmpStart*. Nasleduje popis algoritmu:

1. Vytvorte prioritnú frontu OPEN_1 a vložte do nej štartovací bod s ohodnotením podľa 4.3. Vytvorte prioritnú frontu OPEN_2 a vložte do nej cieľový bod s ohodnotením podľa 4.4. Zostrojte zoznamy CLOSED_1 a CLOSED_2. Priraďte hodnote *tmpGoal* cieľový bod. Priraďte hodnote *tmpStart* štartovací bod. Aktuálny smer prehľadávania je od štartu.

2. Ak je aktuálny smer prehľadávania od štartu, tak vyberte z fronty OPEN_1 prvý bod. V opačnom prípade vyberte prvý bod z fronty OPEN_2. V prípade prázdnej fronty úloha nemá riešenie a prehľadávanie ukončite ako neúspešné.

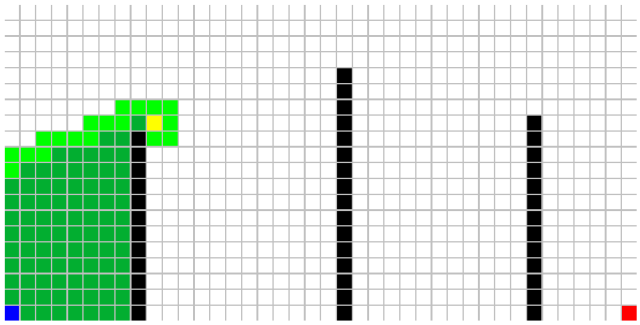
3. Ak je aktuálny smer prehľadávania od štartu a vybraný bod je zároveň bodom *tmpGoal* alebo sa nachádza v zozname CLOSED_2, ukončite prehľadávanie ako úspešné a vráťte výslednú cestu. Ak je aktuálny smer prehľadávania od cieľa a vybraný bod je zároveň bodom *tmpStart* alebo sa nachádza v zozname CLOSED_1, ukončite prehľadávanie ako úspešné a vráťte výslednú cestu. Inak pokračujte.

4. Ak je aktuálny smer prehľadávania od štartu a vybraný bod je zároveň rohový bod, nastavte hodnotu *tmpStart* na vybraný bod, uložte ho do zoznamu CLOSED_1 a prejdite na bod 2. Ak je aktuálny smer prehľadávania od cieľa a vybraný bod je zároveň rohový bod, nastavte hodnotu *tmpGoal* na vybraný bod, uložte vybraný bod do zoznamu CLOSED_2 a prejdite na bod 2. Inak pokračujte.

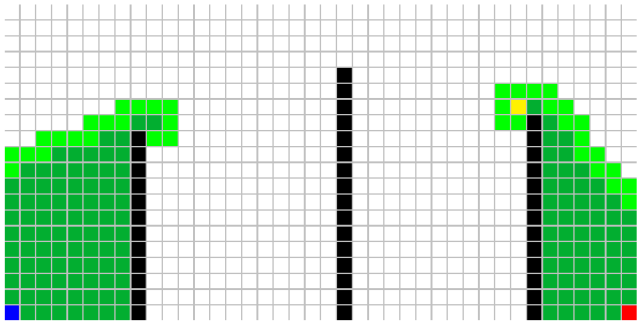
5. Vybraný bod expandujte. Ak je aktuálny smer prehľadávania od štartu, všetkých jeho bezprostredných následníkov ohodnoťte pomocou 4.3 a vybraný bod umiestnite do zoznamu CLOSED_1. Vložte bezprostredných následníkov do fronty OPEN_1 a zaistite, aby v nej zostali len body s najlepším ohodnotením. Ak je aktuálny smer prehľadávania od cieľa, všetkých jeho bezprostredných následníkov ohodnoťte pomocou 4.4 a vybraný bod umiestnite do zoznamu CLOSED_2. Vložte bezprostredných následníkov do fronty OPEN_2 a zaistite, aby v nej zostali len body s najlepším ohodnotením. Vráťte sa na bod 2.

Chovanie algoritmu BMA* v praxi je znázornené na obrázku 4.6. Prvá časť prehľadávania na 4.6a je úplne totožná s algoritmom A*, až kým algoritmus nenarazí na prvý rohový bod (vynačený žltou farbou). V druhej časti prehľadávania na 4.6b je vidieť, že algoritmus zmení smer prehľadávania smerom **od cieľa**, výpočet heuristickej funkcie ale už prebieha voči predtým nájdenému rohovému bodu, až kým sa nezastaví pri novo nájdenom rohovom bode. Tretia časť prehľadávania na 4.6c prebieha zase smerom od počiatočného stavu až do nového rohového bodu. V poslednej asti prehľadávania na 4.6d, ktorá prebieha smerom od cieľa algoritmus narazí na už preskúmaný stav a prehľadávanie sa zastaví ako úspešné.

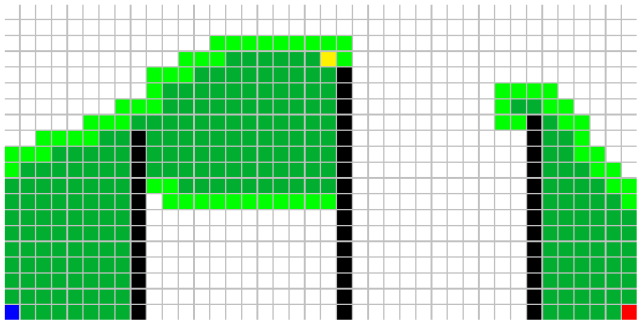
Na príklade z obrázka 4.6 je možné si všimnúť, že nájdená cesta nie je optimálna. Drobná neoptimálnosť vznikla z toho dôvodu, že algoritmus si vymieňa informácie len o prekážkach, na ktoré skutočne narazil a nemôže teda predpokladať a počítať s tým, že sa bude musieť vyhnúť ďalšej prekážke. Pri prehľadávaní však algoritmus nepreskúmal ani zďaleka toľko stavov, ako algoritmus A* (obrázok 4.1) a nájdená cesta je o poznanie kvalitnejšia ako na ostatných algoritmoch (obrázky 4.2 a 4.3).



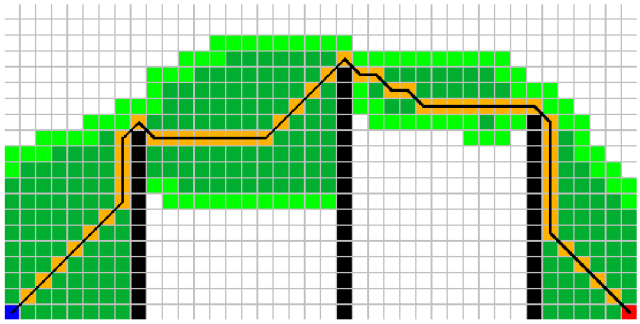
(a)



(b)



(c)



(d)

Obr. 4.6: Postup algoritmu BMA* pri obchádzaní prekážok

Kapitola 5

Implementácia

Táto kapitola obsahuje stručný popis aplikácie implementovanej v jazyku Java a v konkrétne prostredí *NetBeans IDE 8.2*. Aplikácia umožňuje navrhovať a vytvárať množiny bodov s prekážkami a potom cez tieto množiny púšťať algoritmy pre prehľadávanie metrického priestoru uvedené v kapitole 4, pričom toto prehľadávanie je užívateľovi okamžite zobrazené. Prehľadávanie je možné krokovať dopredu, prípade úplne zastaviť.

5.1 Implementácia algoritmov

Algoritmy predstavené v kapitole 4 sú implementované ako metódy triedy **MetricSpace**. Každý algoritmus má svoju inicializačnú metódu a metódu, ktorá predstavuje jeden krok prehľadávania.

Heuristické funkcie, ktoré aplikácia umožňuje použiť sú vlastne metriky definované v časti 3.1, v aplikácií však bola pridaná aj jedna ďalšia heuristika, ktorá bola pre účely práce nazvaná **diskrétna euklidovská heuristika**. Vypočíta sa podľa vzorca:

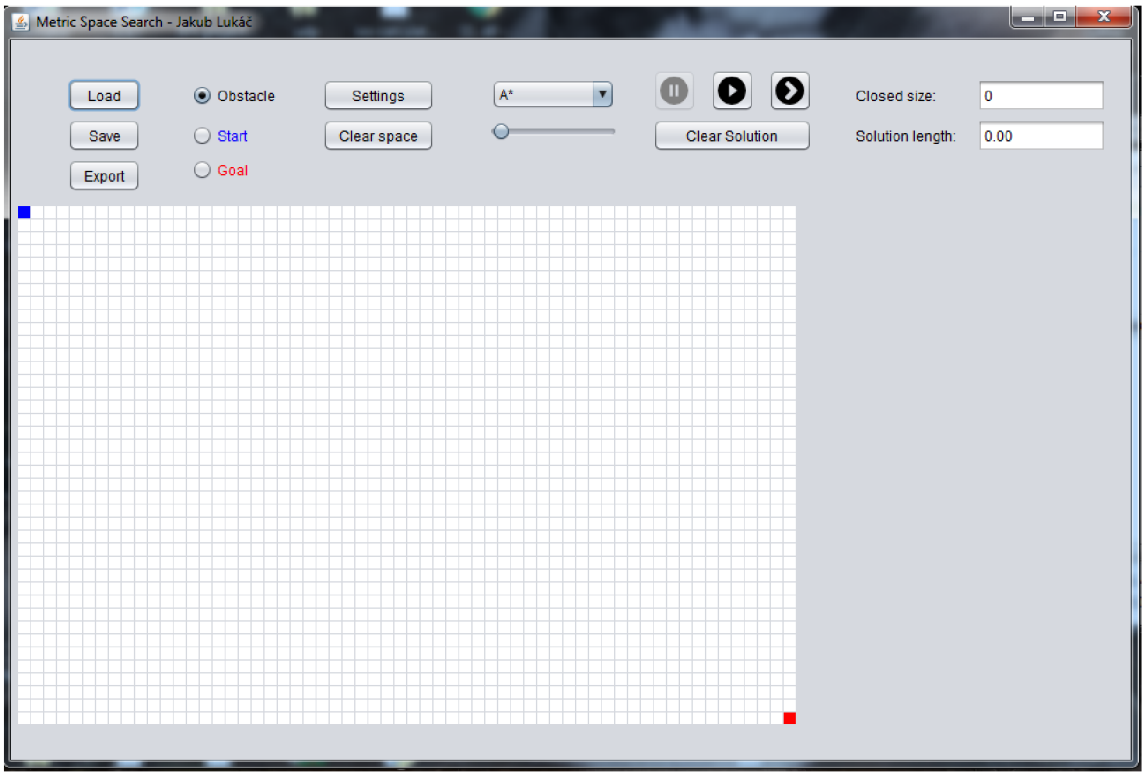
$$h(N) = \sqrt{2} \cdot min(dx, dy) + max(dx, dy) - min(dx, dy) \tag{5.1}$$

kde $dx = |N_x - G_x|$ a $dy = |N_y - G_y|$, pričom G je cieľový bod.

Zoznam closed bol pôvodne implementovaný ako **ArrayList**, v tejto podobe bolo prehľadávanie pomalé. Akonáhle bola spravená zmena closed na **HashSet**, ktorá poskytuje omnoho lepšie výsledky pri operácií *contains*, prehľadávanie sa citeľne zrýchlilo. Fronta open je impementovaná pomocou **PriorityQueue** – v žiadnom preštudovanom literárnom prameni však nebola nájdená zmienka o tom, akým spôsobom má prioritná fronta pre prehľadávanie priestoru ukladať stavy s rovnakým ohodnotením. Toto je možné viacerými spôsobmi: najnovší vkladaný stav s rovnakým ohodnotením vložiť do čela týchto stavov alebo naopak na koniec stavov s rovnakým ohodnotením. Ďalšou možnosťou je ich ukladať náhodným spôsobom a tento spôsob bol využitý v prvotnej verzii aplikácie. Algoritmy si síce stále zachovávali svoje predpísané vlastnosti (optimálnosť, úplnosť), na prvý pohľad však takáto organizácia stavov vo fronte vyzerala pomerne chaoticky. Preto bola využitá trieda **Checker**, ktorá stavy vo fronte open organizuje tak, že posledný pridaný stav s rovnakým ohodnotením ide do ich čela.

5.2 Užívateľské rozhranie aplikácie

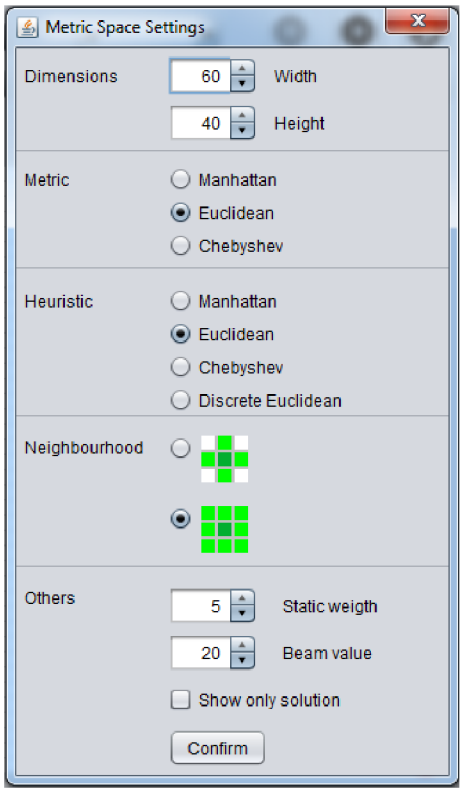
Základom grafického užívateľského rozhrania aplikácie je hlavné okno reprezentované triedou **SpaceJFrame**, pre ktorého ovládacie prvky boli použité komponenty zo štandardnej knižnice Swing. Okno po spustení aplikácie je možné vidieť na obrázku 5.1. Základom okna je horná časť s ovládacími prvkami a plátno, ktoré je potomkom triedy JPanel, čo je jedným z dôvodov nižšej rýchlosti vykresľovania prehľadávania. Po kliknutí na tlačidlo Settings sa otvorí modálne okno (na obrázku 5.2), pomocou ktorého je možné meniť rôzne vlastnosti prehľadávania. Modálne okno je reprezentované triedou **SettingsJDialog**.



Obr. 5.1: Náhľad hlavného okna aplikácie

5.3 Ovládanie

Ovládanie aplikácie je veľmi jednoduché. Prekážky je možné pridávať a odstraňovať ručne klikaním na plátno, ale len v dobe, keď neprebieha žiadne prehľadávanie. Je možné úplne ľuboľne zmeniť polohu počiatočného aj cieľového bodu a to podľa toho, aký typ bodu je zaškrnutý na hlavnom paneli. Vytvorenú množinu bodov je možné uložiť do súboru, prípadne znovu načítať a kedykoľvek meniť. Pokročilejšie nastavenia metrík a heuristík sú možné v modálnom okne **Settings**, kde je možné upraviť veľkosť množiny bodov a to až do veľkosti 100 v horizontálnom smere a 50 vo vertikálnom smere. V modálnom okne je taktiež možnosť úplne vypnúť animáciu prehľadávania, pričom po zapnutí prehľadávania algoritmus zobrazí len výsledok, čo je užitočné v prípade časovo veľmi náročných prehľadávaní. Ďalšie



Obr. 5.2: Náhľad modálneho okna pre nastavenia

zaujímavé nastavenie je typ okolia – **Neighbourhood**, pomocou ktorého je možné si vybrať, či prehľadávanie má nájsť pravouhlú (4) alebo diagonálnu (8) cestu.

Prehľadávanie priestoru sa spustí na hlavnom plátne kliknutím na tlačidlo „Play“, prehľadávanie je možné kedykoľvek prerušiť a znovu spustiť. Program pamätá aj na možnosť nastavenia časového intervalu medzi jednotlivými krokmi prehľadávania. Používateľ má možnosť tento krok nechať prednastavene nulový alebo ho zvýšiť až na 0.2 sekundy (väčší interval už bol aj pri menších priestoroch príliš pomalý).

V akejkoľvek fáze prehľadávania alebo návrhu priestoru je možné obrázok plátna s priestorom exportovať vo formáte **png**, čo je užitočné hlavne vtedy, ak pri prehľadávaní algoritmus nájde zaujímavé riešenie. Všetky obrázky metrických priestorov v tejto práci pochádzajú práve z tejto možnosti aplikácie.

Kapitola 6

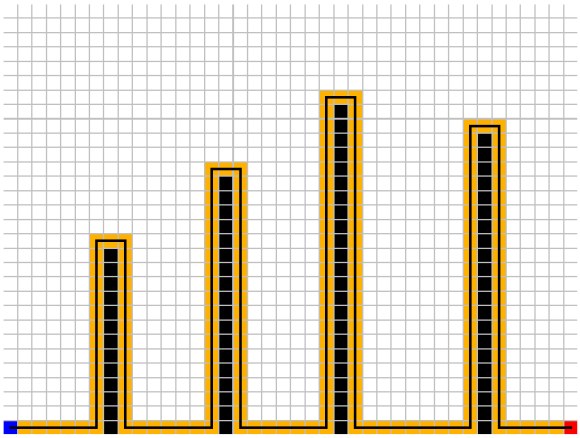
Experimenty

V tejto kapitole sú popísané experimenty na množinách bodov s rozlične rozmiestnenými prekážkami. Množiny bodov boli vytvorené ručne pre potreby tejto práce a sú dostupné v prílohe na pamäťovom médiu ako súbory pre implementovanú aplikáciu z kapitoly 5.

Keďže vytvorená aplikácia umožňuje nad množinou bodov pomocou nastavení vytvoriť veľké množstvo kombinácií metrík a heuristík, rozhodol som sa pre účely tejto kapitoly otestovať len niektoré z nich. Otestované budú množiny bodov s euklidovskou metrikou a euklidovskou a diskrétnou euklidovskou heuristikou, čebyševskou metrikou a čebyševskou heuristikou a ako jediný test pre pravouhlé cesty bude kombinácia manhattanská metrika a manhattanská heuristika. Tieto kombinácie najlepšie otestujú dané množiny bodov, pretože je možné vytvoriť aj také, ktoré nemajú žiadny zmysel, napr. euklidovská metrika a manhattanská heuristika, kde je porušené pravidlo o spodnom odhade pri algoritme A*.

Z dôvodov napísaných pri definícií metódy zjednodušovania priestoru nebude táto metóda zahrnutá do experimentov – pretože je založená na úplne odlišnom princípe, ako ostatné algoritmy, je nemožné dať výsledky tejto metódy do jednotného kontextu s ostatnými metódami.

6.1 Experiment 1



Obr. 6.1: Veľmi neoptimálne riešenie nájdené s greedy search pre prvý experiment

Táto množina bodov je z celej kapitoly najjednoduchšia, medzi počiatkom a cieľom ležia len štyri prekážky. Výsledky testovania sú uvedené v tabuľke 6.1. Hodnoty $X(N)$ uvedené pre každú metódu znamenajú, že nájdené riešenie má dĺžku X a pre jeho nájdenie bolo potrebné preskúmať N bodov. V prípade, že v záhlaví tabuľky pri názve algoritmu nie je explicitne napísaná hodnota, ktorá daný algoritmus upravuje, boli použité prednastavené hodnoty z aplikácie. Každá tabuľka potom obsahuje aj skratky, ktoré značia použitú metriku a heuristiku: v hlavičke M / H znamená metrika a heuristika, E znamená euklidovská (metrika alebo heuristika), D značí diskrétnu euklidovskú heuristiku, CH značí čebyševskú metriku a heuristiku a M značí manhattanskú metriku a heuristiku.

V euklidovskej metrike stanovilo A* optimálne riešenie na $70,36$. Z ostatných algoritmov dopadlo najhoršie greedy search, ktoré sa výrazne odchýlilo od A*, naopak, najrýchlejšie si viedlo BMA*, ktorého výsledky sú veľmi dobré.

Optimum v čebyševskej metrike sa skoro podarilo dosiahnuť BMA*, opäť ako najrýchlejšiemu algoritmu. Veľmi sa neodchýlil ani beam search, ktorý však potreboval pomerne veľké množstvo preskúmaných bodov. Najhorší výsledok dosiahol opäť greedy search, bol však o niečo rýchlejší ako váhované A*.

V manhattanskej metrike dosiahlo optimálneho riešenia ako prvé BMA*. Ostatné algoritmy si neviedli zle, až na greedy search (na obrázku 6.1), kde je vidieť obrovský nedostatok tejto metódy. Môžeme povedať, že v celom experimente si greedy search viedol nedostatočne, naopak BMA* napriek tomu, že výsledky sa od A* odchyľovali, si viedol veľmi dobre.

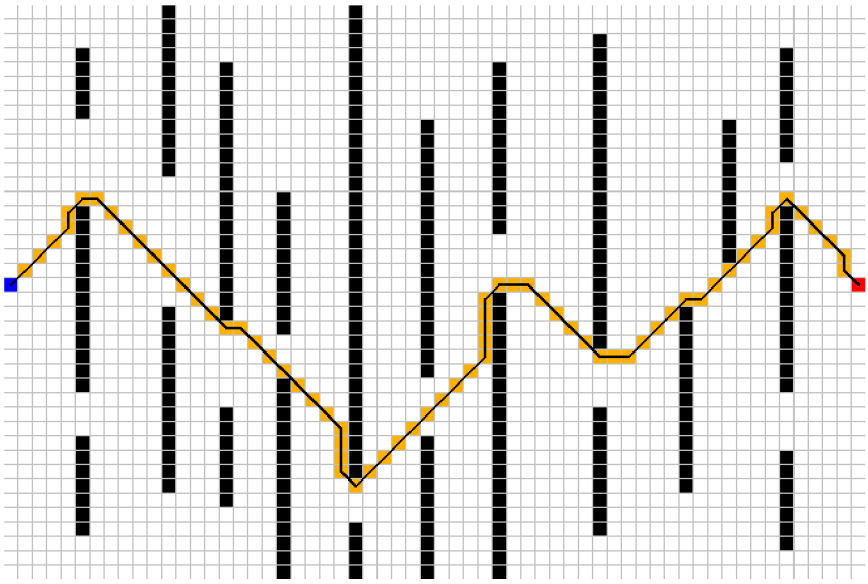
	Metódy				
M / H	**A***	**W. A***	**Greedy**	**Beam**	**BMA***
E / E	70.36 (899)	79.43 (503)	101.08 (445)	77.18 (726)	72.01 (292)
E / D	70.36 (794)	83.08 (526)	101.08 (467)	77.77 (636)	70.94 (289)
CH / CH	60.0 (850)	77.0 (469)	101.0 (422)	63.0 (635)	61.0 (268)
M / M	85.0 (736)	99.0 (562)	189.0 (534)	93.0 (851)	85.0 (298)

Tabuľka 6.1: Tabuľka s výsledkami metód pre prvý experiment

6.2 Experiment 2

Táto množina bodov obsahuje medzi počiatočným a cieľovým bodom celý rad vertikálnych prekážok a existuje teda veľké množstvo možných ciest, ktoré môžu testované algoritmy nájsť. Na testovaní s euklidovskou metrikou (tabuľka 6.2) bolo najzaujímavejšie to, že diskrétna euklidovská heuristika zmenšila počet prehľadaných stavov len pri A*. Váhované A* síce dokázalo nájsť o niečo lepšie riešenie, stále sa však výrazne odchyľuje od optimálneho, ktoré stanovilo A* na hodnotu $90,54$ (na obrázku 6.2). Použitie beam search a metódy BMA* dokonca v tejto heuristike výrazne zhoršilo ako dĺžku nájdeného riešenia, tak aj počet prehľadaných bodov.

Optimálne riešenie v čebyševskej metrike našlo A* v hodnote 69. Všetky ostatné algoritmy sa od optimálnej hodnoty odchýlili až drasticky, beam search s hodnotou $\beta = 20$ dokonca skončil bez akéhokoľvek výsledku. Algortimus BMA* našiel lepšie riešenie ako greedy či váhované A*, no musel kvôli tomu prehľadať skoro dvojnásobné množstvo bodov ako A*, pričom riešenie je stále dosť zlé.



Obr. 6.2: Optimálne riešenie nájdené s A* pre druhý experiment

V manhattanskej metrike si viedli všetky algoritmy prekvapivo skoro rovnako až na beam search, ktorý opäť nenašiel žiadne riešenie. Optimálnu hodnotu 121 najrýchlejšie dosiahlo greedy search.

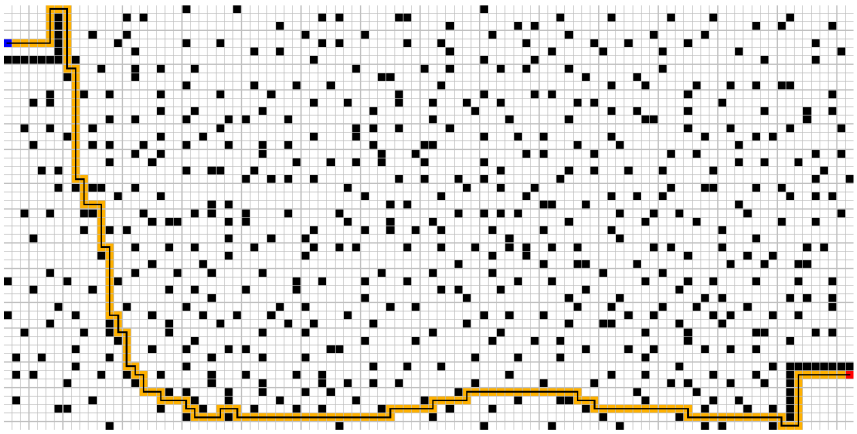
M / H	**Metódy**				
	A*	**W. A***	**Greedy**	**Beam**	**BMA***
E / E	90.54 (1381)	103.08 (203)	105.81 (159)	98.05 (878)	95.23 (875)
E / D	90.54 (1339)	99.81 (224)	105.81 (184)	115.81 (1111)	102.88 (1034)
CH / CH	69.0 (1041)	113.0 (209)	113.0 (131)	- (761)	97.0 (1936)
M / M	121.0 (1437)	121.0 (465)	121.0 (344)	- (747)	123.0 (1427)

Tabuľka 6.2: Tabuľka s výsledkami metód pre druhý experiment

6.3 Experiment 3

Táto množina bodov neobsahuje spojené prekážky, ako v predchádzajúcom experimente, ale veľké množstvo malých prekážok, ktoré sú do priestoru rozmiestnené viac-menej náhodne. Množina bola vytvorené hlavne pre otestovanie algoritmu BMA*, pretože takto rozmiestnené prekážky obsahujú veľké množstvo rohových bodov (a tým pádom aj veľké množstvo zmien smeru prehľadávania).

Optimálnu dĺžku riešenia v euklidovskej metrike stanovilo A* na $123,44$ (tabuľka 6.3), použitie diskrétnej euklidovskej heuristiky v tomto prípade dokázalo riešenie urýchliť. V počte prehľadaných bodov je na tom A* najhoršie – tu si je možné všimnúť, aký veľký rozdiel môže spôsobiť vynechanie zložky $g(n)$ v ohodnocovacej funkcii. Greedy search dokázalo nájsť celkom prijateľné riešenie skoro devätnásťkrát rýchlejšie. Najlepšie vyšli výsledky pravdepodobne algoritmu BMA*, ktorého výsledky sa od optimálneho odchyľovali len veľmi mierne, rýchlosťou bol však porovnateľný s greedy search.



Obr. 6.3: Mierne neoptimálne riešenie nájdené s BMA* v treťom experimente

Podobne, ako v prvom experimente, aj teraz bolo v čebyševskej metrike BMA* najpomalšie, tentokrát však aj s najhorším riešením. Ostatné metódy sa od optimálneho riešenia príliš neodchyľovali, beam search dokonca našiel optimálne rovnakým spôsobom, ako A*.

V manhattanskej metrike našlo optimum len A* s hodnotou 158. K nájdeniu optimálneho riešenia však bolo treba preskúmať obrovské množstvo bodov. Riešenie porovnateľnej kvality našlo BMA* (na obrázku 6.3) skoro devätnásťkrát rýchlejšie.

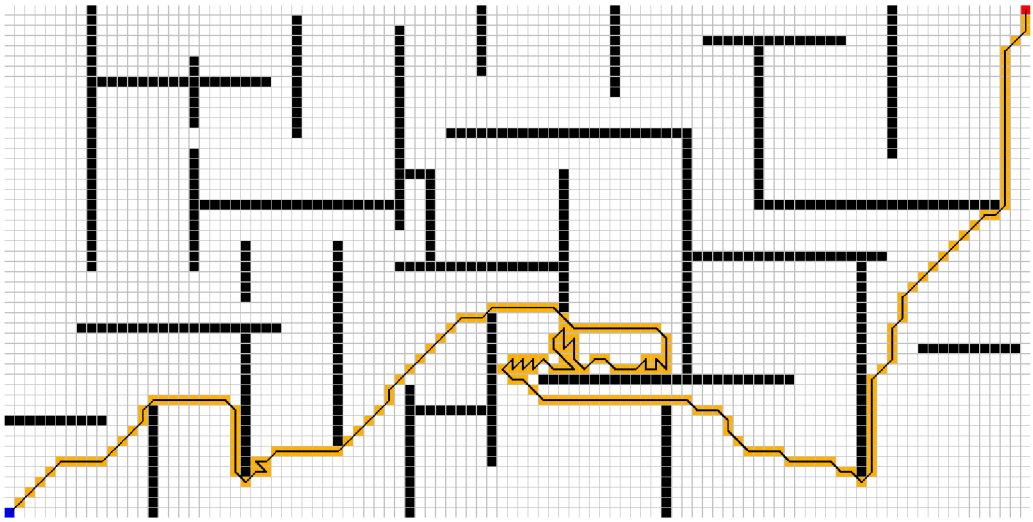
M / H	**Metódy**				
	A*	**W. A***	**Greedy**	**Beam** $\beta = 50$	**BMA***
E / E	123.44 (3108)	130.02 (171)	132.51 (166)	138.12 (2784)	126.02 (216)
E / D	123.44 (2740)	132.51 (173)	132.51 (168)	132.61 (2180)	128.61 (160)
CH / CH	99.0 (110)	102.0 (104)	102.0 (104)	99.0 (110)	106.0 (137)
M / M	158.0 (4304)	178.0 (284)	176.0 (270)	166.0 (3433)	166.0 (229)

Tabuľka 6.3: Tabuľka s výsledkami metód pre tretí experiment

6.4 Experiment 4

Táto množina bodov už pripomína bludisko – nejedná sa o nijako náhodne rozmiestnené body, obsahuje už celkom kompikované steny a množstvo „zákutí“, kam môže pri prehľadávaní algoritmus vstúpiť.

Domnievam sa, že takto rozmiestnené prekážky sú presne ten typ úloh, kde sa ukážu nedostatky rýchlejších algoritmov. Greedy search v oboch testoch na euklidovskej metrike našlo riešenie najrýchlejšie (v tabuľke 6.4), od optimálnej hodnoty $158,17$ je však horšie $1,5$krát. Algoritmus A* opäť prehľadal najväčšie množstvo bodov. Zaujímavé je, že všetky ostatné algoritmy sa tiež optimálnemu riešeniu ani nepriblížili – najbližšie sa k nemu dostal BMA* s euklidovskou heuristkou, potreboval na to však prekonať skoro 2400 bodov. Ako zaujímavosť uvediem výsledok beam search v diskrétnej euklidovskej metrike (na obrázku 6.4), kde nájdené riešenie vyzerá celkom prijateľne až na jednu hrubú neoptimalitu približne v strede priestoru.



Obr. 6.4: Neoptimálne riešenie nájdené algoritmom beam search v štvrtom experimente

V čebyševskej metrike nastala rovnaká situácia ako v prvom experimente, k optimálnej hodnote sa nepriblížil nijaký algoritmus okrem A*. Riešenia všetkých ostatných metód sa príliš odchyľovali, spomeniem akurát fakt, že greedy search našlo riešenie aspoň skoro štyrikrát rýchlejšie.

V manhattanskej metrike nastal opäť nežiadaný jav – beam search nenašlo žiadne riešenie. Preto som aspoň pre tento konkrétny test zvýšil hodnotu β na 30, pri ktorom už beam search našiel veľmi dobrý výsledok. Od optimálnej hodnoty 194 sa odchyľuje len mierne a bol nájdený v porovnaní s A* aj celkom rýchlo. Naopak, o trochu pomalšie než beam search si viedlo BMA*, nájdené riešenie je ale v tomto teste zďaleka najhoršie.

	Metódy				
M / H	**A***	**W. A***	**Greedy**	**Beam**	**BMA***
E / E	158.17 (3943)	185.68 (952)	241.87 (681)	184.27 (2009)	175.14 (2361)
E / D	158.17 (3622)	189.44 (946)	241.87 (661)	218.99 (2292)	183.48 (1520)
CH / CH	130.0 (3456)	170.0 (1131)	203.0 (901)	172.0 (2231)	170.0 (2228)
M / M	194.0 (3314)	248.0 (819)	248.0 (681)	198.0 (2008)	260.0 (2421)

Tabuľka 6.4: Tabuľka s výsledkami metód pre štvrtý experiment

6.5 Experiment 5

Táto testovacia množina s prekážkami sa dá považovať za regulárne bludisko a je riadnou výzvou pre akýkoľvek prehľadávací algoritmus. V euklidovskej metrike opäť najmenej kvalitné riešenie našiel algoritmus greedy, ktorý ukazuje, že pre takto zložito rozmiestnené prekážky je absolútne nevhodný. Riešenie výraznejšie sa odchyľujúce od optima 16.92 našiel BMA* (v tabuľke 6.5), ktorý bol v oboch prípadoch približne o polovicu rýchlejší, ako A*. V diskrétnej euklidovskej metrike naopak úplne zlyhal beam search, ktorý ukazuje, že by mal byť použitý len v skutočne opodstatnených prípadoch.

V čebyševskej metrike opäť beam search nenašiel žiadne riešenie, BMA* našiel relatívne prijateľné riešenie približne trikrát rýchlejšie, ako A*. Váhované A* poskytlo o niečo lepší výsledok ako greedy, obidve tieto metódy sa však príliš vzdialili od optima.

V manhattanskej metrike naopak prekvapilo beam search, riešenie s dĺžkou 208 je v tomto teste druhé najlepšie. Ostatné algoritmy predstihli beam search a A* v rýchlosti, dĺžka ich riešení je však neprijateľná. Zaujímavosťou je, že v celom experimente pôsobilo váhované A* ako kompromis medzi A* a greedy search – bolo pomalšie než greedy, ale poskytlo lepšie výsledky.

	Metódy				
M / H	**A***	**W. A***	**Greedy**	**Beam** $\beta = 40$	**BMA***
E / E	163.92 (2780)	199.62 (537)	221.28 (412)	184.89 (2536)	180.11 (1306)
E / D	163.92 (2487)	200.25 (511)	221.28 (385)	- (3053)	182.41 (1106)
CH / CH	137.0 (2594)	172.0 (562)	184.0 (462)	- (3169)	145.0 (878)
M / M	202.0 (2373)	242.0 (644)	262.0 (553)	208.0 (2024)	254.0 (1889)

Tabuľka 6.5: Tabuľka s výsledkami metód pre piaty experiment

6.6 Zhodnotenie výsledkov

Hoci boli v priebehu experimentov popísané výsledky a bolo upozornené na niekoľko zaujímavých javov, v tejto časti je vhodné zhrnúť vedomosti, ktoré sme sa dozvedeli o jednotlivých metódach na prevedených experimentoch. Vďaka vlastnostiam implementovanej aplikácie môže mať množina bodov s prekážkami maximálne 5000 bodov, čo nie je nijaká závratná hodnota a nemusí úplne stačiť k ťažkému otestovaniu všetkých algoritmov, ktorých dobré výsledky sa môžu prejavovať až pri oveľa väčšom počte bodov.

Ako vyplýva z definície metódy A* a toho, že sme použili prípustné heuristické funkcie, algoritmus vždy našiel optimálne riešenie. Nemusí to však nutne znamenať, že je to jediný prakticky použiteľný algoritmus. Vo väčšine experimentoch práve A* skončilo prehľadávanie ako posledné a ak je prioritou riešiteľa to, aby metóda našla riešenie čo najrýchlejšie, A* nemusí túto podmienku spĺňať.

Testovať váhované A* pre veľké hodnoty ε nedáva veľký zmysel. Pri veľkostiach testovaných množín sa už prehľadávanie s hodnotami ε väčšími ako 10 veľmi podobalo výsledkami na greedy search. Hodnoty menšie ako 4 sa naopak veľmi približovali výsledkami k A*, preto bolo v experimentoch rozhodnuté testovať na prednastavenej hodnote $\varepsilon = 5$. Riešenie môže naberať pomerne dobré hodnoty, často však tento algoritmus pôsobí ako kompromis medzi A* a greedy search ako pre časové, tak pre pamäťové vlastnosti. Ak nám pri greedy search vadí vysoká odchylka od optima, ale chceme sa priblížiť k jeho časovým vlastnostiam, je použitie váhovaného A* prijateľné.

Najväšou nevýhodou oproti ostatným testovaným algoritmom pri beam search je jeho neúplnosť. Prakticky pri všetkých testovaných príkladoch bolo beam search pomalšie, ako BMA*. Zahadzovanie stavov s menej sľubným ohodnoteným môže zmenšiť pamäťové nároky metódy, nemá však zmysel túto hranicu ťahať do úplného minima. Pri hodnote $\beta = 1$ si algoritmus uchováva v open prakticky len jeden stav, výsledky sú však mimoriadne zlé (ak vôbec v takom prípade riešenie našlo). Používať beam search má zmysel pravdepodobne len pre vyššie hodnoty β, kde sa riešenie už tak nelíši od A*.

Obecne môžeme konštatovať, že výsledky metódy BMA* nie sú špatné. Algoritmus vo väčšine prípadov predstihol A* v rýchlosti, ale presne optimálne riešenie našiel len veľmi zriedka. Nájdené riešenie sa väčšinou neodchyľovalo takým drastickým spôsobom, ako greedy. To, že si BMA* v prvom experimente viedlo najlepšie nie je nijak prekvapivé – presne s ohľadom na takýto typ úlohy bol algoritmus navrhnutý a predpokladám, že si podobné výsledky zachová aj v podobnom type úloh v oveľa väčšej množine bodov. Čo sa týka pamäťových požiadavkoch, tak tie v prevedených algoritmoch boli podobné metóde A*, hlavný rozdiel medzi nimi robí o niečo komplikovanejšia implementácia BMA*, ktorá používa dve fronty open a dva zoznamy closed.

Môžeme uvažovať aj o spôsobe, akým algoritmus BMA* vylepšiť, resp. zmeniť jeho vlastnosti. Predstavoval by som si to tak, že upravená verzia BMA* by nemusela zmeniť smer prehľadávania po každom nájdenom rohovom bode. Pred začiatkom prehľadávania by mohlo byť určené, ktoré rohové body je potrebné ignorovať – mohlo by to byť určené napr. ich vzdialenosťou od počiatku, od cieľa, prípade tým, aký typ ďalších bodov sa nachádza v ich okolí. Potom by sa teoreticky dalo zabrániť neefektívne veľkému množstvu zmien smerov, aké nastávajú v experimente 6.3, ale na druhú stranu preskakovať len v určitých prípadoch.

Kapitola 7

Záver

Hlavným cieľom tejto práce bolo vytvoriť aplikáciu pre prehľadávanie metrického priestoru s prekážkami. V rámci práce boli predstavené základné algoritmy pre prehľadávanie stavového priestoru. Spolu s podrobnejšie prezentovanými efektívnymi metódami boli predstavené dva nové algoritmy – prvý z nich sa snaží zrýchliť tradičný prístup tým, že priestor s prekážkami prevedie na zjednodušený graf a tento graf je následne prehľadávaný. Transformácia na graf však môže byť značne neefektívna, hlavne ak neexistuje nijaká predstava o tom, ako sú v danom priestore rozložené prekážky. Preto má prvá metóda len teoretické použitie. Druhým predstaveným algoritmom je spojenie obojsmerného prehľadávania a A*, ktorý bol nazvaný BMA*. BMA* strieda smer prehľadávania podľa toho, či z druhej strany nenarazil na bod, cez ktorý je možné obísť prekážku. Algoritmy boli implementované do aplikácie v jazyku Java.

Pre vytvorenú aplikáciu existuje množstvo možných vylepšení. Aktuálna podoba aplikácie nedovoľuje vytvoriť priestor s viac ako 5000 bodmi hlavne z dôvodu toho, aby výsledný priestor zobrazený užívateľovi bol dostatočne jasný a aby postup algoritmov pri prehľadávaní bol dostatočne demonštratívny. V upravenej podobe by mohla aplikácia dokázať prehľadávať oveľa väčšie množstvo bodov. Čo sa týka ovládania, vkladanie prekážok do priestoru je veľmi jednoduché a všetko musí užívateľ spraviť sám pomocou myši. V upravenej aplikácií si dokážem predstaviť nástroj, ktorý by dokázal náhodne vygenerovať rôzne druhy prekážok. Pri vypnutí zobrazovania prehľadávania je možné si všimnúť toho, že samotné prehľadávanie je veľmi rýchle. Keď je však táto možnosť zapnutá, aj pri nulovej nastavenej prestávke medzi jednotlivými krokmi prehľadávania toto trvá pri väčších priestoroch pomerne dlhú dobu. Stálo by za to uvažovať nad tým, ako vykresľovanie prehľadávaných bodov zrýchliť a zjednodušiť.

Ak berieme do úvahy kvalitu výsledkov prehľadávania v uvedených experimentoch, tak sa javí ako najlepšia metóda **A***. Optimálnosť môže byť zásadný požiadavok a žiadna iná testovaná metóda ho nebola schopná plniť obecne, aj keď sa našlo niekoľko príkladov, kedy bolo optimum nájdené. V časových a pamäťových vlastnostiach si najlepšie viedol algoritmus **greedy search**. Vo väčšine prípadov bolo riešenie touto metódou nájdené najrýchlejšie a tak isto bolo takmer vždy toto riešenie aj najhoršie. Použitie metódy greedy search je teda prijateľné len vtedy, ak sú požiadavky na nájdenie akéhoľvek riešenia, bez ohľadu na jeho kvalitu. Navrhnutý algoritmus **BMA*** je vhodný skôr pre priestory s jednoduchými prekážkami – pri prehľadávaní zložitých bludísk sa riešenie v závislosti na zvolenej heuristickej funkcii a na použitej metrike môže od optimálneho výrazne líšiť. V prevedených experimentoch však bolo spravidla rýchlejšie ako A*.

Literatúra

[1] Kovár, M.: *Diskrétní matematika (informační technologie).* FEKT VUT v Brně, 2014.

[2] Wilt, C.; Thayer, R.; Ruml, W.: A Comparison of Greedy Search Algorithms. In *Proceedings of the Third Annual Symposium on Combinatorial Search (SOCS-10)*, AAAI Press, 2010, s. 129–136.

[3] Zbořil, F. V.; Zbořil, F.: *Základy umělé inteligence IZU: Studijní opora.* FIT VUT v Brně, 2012.

[4] Zezula, P.; Amato, G.; Dohnal, V.; aj.: *Similarity Search: The Metric Space Approach.* Springer US, 2006, ISBN 0-387-29146-6.

[5] Zhang, W.: *State-Space Search: Algorithms, Complexity, Extensions and Applications.* Springer-Verlag New York, 1999, ISBN 978-0-387-98832-0.

Príloha A

Obsah priloženého DVD

Súčasťou DVD priloženého k práci sú:

- Zdrojové súbory tejto bakalárskej práce v LaTeXe a táto práca vygenerovaná v **.pdf**,
- Zdrojové súbory aplikácie pre prehľadávanie metrického priestoru ako projekt do prostredia NetBeans,
- Spustiteľný **.jar** súbor realizačného výstupu práce,
- Testovacie množiny bodov, ktoré boli použité pre experimenty s algoritmami.