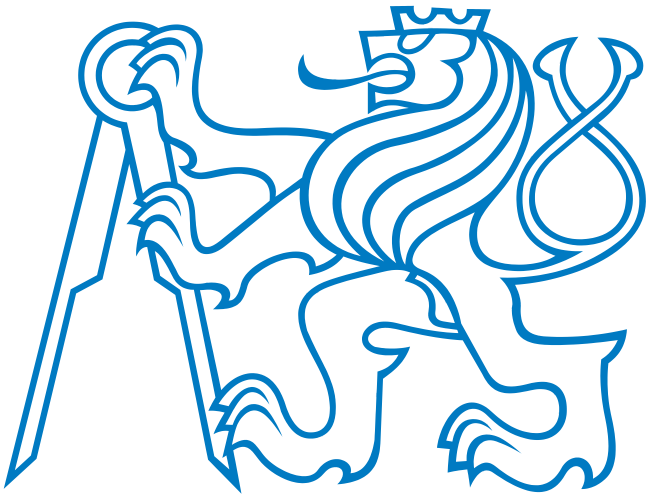
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA ELEKTROTECHNICKÁ

BAKALÁŘSKÁ PRÁCE



Vladislav Vávra

Hledání nejkratší cesty v polygonální doméně

Katedra kybernetiky

Vedoucí bakalářské práce: **Ing. Jan Faigl, Ph.D.**

Praha, 2011

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, SW, projekty atd.) uvedené v přiloženém seznamu.

V Praze dne . . . . . .

. . . . . .
podpis

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Vladislav V á v r a

Studijní program: Elektrotechnika a informatika (bakalářský), strukturovaný

Obor: Kybernetika a měření

Název tématu: Hledání nejkratších cest v polygonální doméně

Pokyny pro vypracování:

1. Seznamte se s úlohou hledání nejkratších cest v polygonální doméně [1, 2, 3].
2. Seznamte se s algoritmy hledání nejkratší cesty [4, 5, 2, 1, 7]
3. Seznamte se s knihovnou CGAL [6] a podpůrnými knihovnami skupiny IMR pro plánovací algoritmy.
4. Implementujte vybrané algoritmy hledání nejkratší cesty v polygonální doméně.
5. Implementace porovnejte v úloze plánování cesty přes více cílů řešené samo-organizující se sítí [8].

Seznam odborné literatury:

[1] Lavalle, S. M.: Planning Algorithms. Cambridge University Press, May 2006.
[2] Goodman, J. E. and O'Rourke, J. editors: Handbook of Discrete and Computational Geometry. CRC Press LLC, Boca Raton, FL, 2004.
[3] Hershberger, J. and Suri, S: An Optimal Algorithm for Euclidean Shortest Paths in the Plane. SIAM J. Comput., 28(6):2215–2256, 1999.
[4] Hua Guo, Anil Maheshwari, and Jörg-Rüdiger Sack: Shortest path queries in polygonal domains. In Rudolf Fleischer and Jinhui Xu, editors, Algorithmic Aspects in Information and Management, volume 5034 of Lecture Notes in Computer Science, pages 200–211. Springer Berlin / Heidelberg, 2008.
[5] Srinivasan V. and Nackman L.R. : Voronoi diagram for multiply-connected polygonal domains i: algorithm. IBM Journal of Research and Development, 3(31):361–372, 1989.
[6] CGAL - Computational Geometry Algorithms Library. http://www.cgal.org. citováno říjen, 2004.
[7] Faigl, J.: Approximate Solution of the Multiple Watchman Routes Problem with Restricted Visibility Range. IEEE Transactions on Neural Networks, 21(10):1668–1679, 2010.
[8] Faigl, J.: Multi-Goal Path Planning for Cooperative Sensing. PhD thesis, Czech Technical University in Prague, 2010.

Vedoucí bakalářské práce: Ing. Jan Faigl, Ph.D.

Platnost zadání: do konce zimního semestru 2011/2012

L.S.

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 3. 2. 2011

Abstrakt

Bakalářská práce se zabývá problémem hledání nejkratších cest v polygonální doméně. Studovaný problém je motivován úlohami plánování cesty pro mobilní roboty v geometrické reprezentaci pracovního prostředí robotu. V práci jsou nejdříve představeny základní přístupy řešení problému hledání nejkratší cesty. Na přehled přístupů navazuje detailní popis algoritmu, který je inspirován optimálním algoritmem hledání nejkratší cesty, jež využívá teselaci volného prostoru založenou na Voroného diagramu. Tato teselace rozděluje volný prostor na trojúhelníky a jednoduché polygony a tvoří základní urychlující strukturu pro dotazy na nejkratší cesty mezi dvěma libovolnými body ve volném prostoru. Vlastní konstrukce nejkratší cesty využívá přepočítaného grafu viditelnosti a tzv. kritických bodů určených z expanze funnelů. Tento algoritmus byl implementován a v práci jsou uvedeny výsledky ověřujících experimentů správnosti implementace a skutečné výpočetní náročnosti. Mimoto je v práci uvedeno porovnání výpočetní náročnosti algoritmu s přístupy založenými na konstrukci grafu viditelnosti a také aproximačním algoritmem poskytujícím odhad nejkratší cesty. V závěru experimentů je pak posouzen vliv výpočtu nejkratší cesty na řešení úlohy obchodního cestujícího technikou samoorganizující se neuronovou sítí.

Abstract

The thesis deals with the problem of the shortest path queries in the polygonal domain. The studied problem is motivated by the path planning for a mobile robot and geometric representation of the robot workspace. First, several approaches are briefly introduced including exact and approximate solutions. A selected exact approach is described in detail way. This approach utilizes a tessellation based on Voronoi diagram, which tessellates free space into triangles and simple polygons, providing a structure for fast shortest path queries. Based on so-called funnel expansion the critical points are determined, which are then used in shortest path construction from the visibility graph. The algorithm has been implemented and experimental results proving correctness of the implementation and the computational requirements are presented in the thesis. Besides, a comparison of the computational requirements of approaches based on the visibility graphs and approximate solution are presented as well. In addition, the exact shortest path is applied in the self-organizing map approach for the traveling salesman problem.

Děkuji všem, kteří mě podporovali během této práce a to především panu Faiglovi za jeho cenné rady a ochotu kdykoliv pomoci. Dále bych pak rád poděkoval svým rodičům a přátelům, kteří to celé se mnou vydrželi …

Obsah

Seznam tabulek

Seznam obrázků

Seznam algoritmů

Kapitola 1

Úvod

Problém plánování cesty pro autonomní pohyb robotu se historicky objevil u prvního autonomního robotu Shakey [9] vyvinutém v roce 1972 v Standford Research Institute. Shakey byl například schopen v místnostech propojených chodbami samostatně rozsvěcet a zhasínat světla. Robot potřeboval ke své činnosti spousty vybavení a to jak hardwarového, tak softwarového. Jedním z algoritmů, které musel robot „znát" je algoritmus pro hledání cesty v pracovním prostředí. Hlavním požadavkem na cestu je bezkoliznost, která má zajistit bezpečný pohyb robotu. Bezkolizních cest může být ovšem nekonečně mnoho, a proto jsou uvažována další kritéria. Takovým kritériem může být například délka cesty, kdy je cílem plánování optimalizovat délku na minimální možnou. V projektu Shakey se poprvé použilo řešení tohoto problému s využitím grafu viditelnosti, tj. geometrický přístup založený na prohledávání polygonální domény.

Pracovní prostředí robotu může být popsáno několika způsoby, tři nejběžnější jsou následující. První způsob je rozdělení prostoru rovnoměrnou mřížkou na jednotlivé oblasti. U tohoto přístupu je možné cestu nalézt algoritmem prohledávání grafů. Dalším přístupem je topologická reprezentace pracovního prostoru, kde jsou do diskrétního grafu uloženy pouze jednotlivé významné objekty prostředí (například dveře, chodba, ...), které jsou spojeny hranou právě tehdy, existuje-li mezi nimi průjezdná cesta. Dalším přístupem je geometrická reprezentace prostoru, kdy jednotlivé překážky v prostředí představují jednoduché polygony, mezi kterými se robot pohybuje. Na tomto přístupu je založen algoritmus, který je předmětem této práce.

Ve studovaném problému se prostor skládá pouze z překážek, hranice prostoru a volného prostoru mezi hranicí a překážkami. Robot se může pohybovat pouze ve volném prostoru. Otázkou zůstává, jak v takto popsaném prostředí robot reprezentovat a jak zohlednit jeho pohyblivost. Existují roboty, které se mohou pohybovat všemi směry, jiné ovšem mohou být omezeny například možností natočení. Dále je nutné zohlednit tvar a rozměry robotu. Řešení hledání cesty v geometrickém prostředí s uvedenými vlastnostmi robotu představuje stále velmi komplikovanou úlohu. Z tohoto důvodu je pro reprezentaci robotu využit tzv. bodový model robotu, což je reprezentace, ve které robot představuje pouze jeden bod v prostoru. Toto je velmi významné zjednodušení, které dosavadní obtížnost problému přesouvá do kategorie optimálních algoritmů, tj. problémů řešitelných v polynomiálním čase. Otázkou zůstává, zda se může takto reprezentovaný robot dotknout hranic překážek, tj. otázka, zda překážky představují otevřenou nebo uzavřenou množinu. Jelikož je robot reprezentován pouze bodem, nemá příliš smysl uvažovat případ, kdy by se překážky dotknout nemohl. Proto lze uvažovat o překážkách jako o ote-

vřené množině a o volném prostoru jako uzavřené. Otázku kolize robotu v takto popsaném problému je možné řešit například vhodným zvětšením rozměrů překážek využitím Minkowského sumy a diskového modelu robotu. Uvedená reprezentace pracovního prostředí robotu je nazývá polygonální doména a textu je pro ni též označována jako mapa.

1.1 Algoritmy hledání nejkratších cest

Všechny algoritmy pro hledání nejkratších cest v polygonální doméně lze rozdělit na exaktní a aproximační. Aproximační algoritmy používají pro nalezení cesty různé metody odhadu. Z tohoto důvodu je možné, že jimy nalezená cesta nebude optimální, bude ovšem nalezena v poměrně krátkém čase. Příklad takového algoritmu je uveden v [4], jehož složitost je v nejhorším případě $O(n)$, ale praktická složitost je mnohem menší. Exaktní algoritmy na rozdíl od aproximačních používají postupy, které vedou vždy k nalezení nejkratší cesty. Časová náročnost takovýchto algoritmů bývá větší než u algoritmů pro odhad cesty. Předmětem této práce je exaktní algoritmus, a proto je dále uveden krátký úvod do variant exaktních algoritmů pro hledání nejkratší cesty.

1.1.1 Exaktní algoritmy pro hledání nejkratších cest

Existuje několik přístupů pro hledání nejkratší cesty. Tyto přístupy lze dělit podle několika kritérií. První kritérium popisuje složitost podpůrné datové struktury, která je pro hledání nejkratší cesty využita. Nejjednodušším přístupem je graf viditelnosti, který pouze popisuje, které vrcholy jsou vzájemně viditelné a jak daleko se od sebe nachází. Existuje několik variant tohoto přístupu. Jednou z nich je například algoritmus konstrukce viditelných vrcholů z jednoho bod. Tento přístup má složitost nalezení viditelných bodů $O(n\log(n))$ [2], kde n je počet bodů, pro které je graf počítán. Tento přístup je dále v textu značen jako *visll*. Dále existuje alogritmus (zde značen jako *visrot*), který konstruje viditelné body pro všechny vrcholy (mapy i požadované koncové body cesty) najednou a jehož složitost je $O(n^2)$ [8], kde n je celkový počet vrcholů. Hledání cesty v obou přístupech je pak provedeno algoritmem hledání cesty v grafu (v případě *visll* je nutné ještě předpočítat úplný graf viditelnosti pro celou mapu), například užitím Dijkstrova algoritmu se složitostí $O(|e|+n\log n)$, kde $|e|$ celkový je počet hran grafu a n počet jeho vrcholů (včetně vrcholů reprezentující koncové body hledané cesty). Dále existují přístupy, které využívají sofistikovanější datové struktury, a mají proto lepší složitost. Jejich stručný přehled je uveden v následujícím odstavci.

Dalším kritériem pro dělení exaktních algoritmů je podle počtu bodů, které mohou být volitelné pro hledání nejkratší cesty. První skupinou jsou algoritmy typu *single-source-query* SSQ (v češtině jedno zdrojové dotazy). Tyto algoritmy předpokládají, že jeden z dotazovaných bodů v prohledávaném prostoru bude neměnný a k tomuto bodu se provede přípravný výpočet. Jedním z příkladů takového algoritmu je například Shortest Path Map [6] (mapa nejkratších cest), který rozdělí volný prostor na takové oblasti, pro které vždy jeden vrchol každé oblasti leží na nejkratší cestě k cíli. Složitost výpočtu přípravné datové struktury je $O(n\log n)$ a složitost jednoto dotazu na vzdálenost je pak $O(\log n)$, kde n je počet vrcholů mapy. Alternativním příkladem takovéhoto algoritmu je již uvedený graf viditelnosti s předpočítaným grafem pro vybraný bod a všechny vrcholy mapy.

Další kategorií exaktních algoritmů jsou *all-pair-query* APQ algoritmy, kde oba body mezi kterými volíme nejkratší cestu mohou být libovolné. Takovýto algoritmus předsta-

vil Chen v [3]. Složitost přípravy je $O(n^2 \log n)$ a složitost dotazů na vzdálenost je pak $O(min(Q_a, Q_b) \log n)$, kde Q_a (Q_b) je počet vrcholů viditelných z a (b). Nevýhoda tohoto algoritmu je jeho necitlivost na počet překážek. Tento nedostatek odstraňuje algoritmus uvedený pány H. Guem, A. Maheshvrcholy warim a J. Sackem v [5]. Složitost přípravy je $O(n^2 \log n)$ a složitost jednotlivých dotazů pak $O(h \log n)$, kde h je počet překážek. Tento algoritmus je inspirací pro algoritmus popsaný v této práci.

1.2 Optimální APQ algoritmus

Tento algoritmus je založen na hledání tzv. kritických bodů, tj. takových bodů, které leží na vrcholu mapy a prochází jimy optimální cesta. Po nalezení množiny možných kritických bodů se provede vyhodnocení, které z možných kritických bodů jsou skutečně kritické a na základě dotazu v SPM je nalezena nejkratší cesta mezi těmito body a koncovými body cesty. Kritické body jsou přímo viditelné z počátečního, respektive cílového bodu hledání, a tak cesta nalezená užitím SPM je pouze rozšířena o hledané body.

Část algoritmu hledající kritické vrcholy je založena na propagaci funnelů (množina všech nejkratších cest z bodu na úsečku). Pro tuto propagaci se využívá teselace volného prostoru (rozdělení volného prostoru na oblasti) založena na Voroného diagramu. Teselace rozdělí volný prostor na trojúhelníky a jednoduché polygony. Výhodou tohoto rozdělení je jeho citlivost na množství překážek a jednoznačnost pro zadanou mapu. V tomto rozdělení se dále určí pro každý jednoduchý polygon množina všech nejkratších cest spojující brány (dělící úsečka dvou oblastí teselace) tohoto polygonu. S využitím tohoto rozdělení se pak vyhledají kritické body. Dále algoritmus předpočítá mapu nejkratších cest kterou využije na určení, jak kritické body propojit nejkratší cestou.

Algoritmus popsaný v [5] využívá pro svůj běh a pro přípravné výpočty velké množství datových struktur jejichž implementace není triviální. Z tohoto důvodu byla použita implementace některých geometrických konstrukcí knihovny CGAL [1] a byla zavedena určitá zjednodušení implementace, především v SPM, místo které byly použiti grafy viditelnosti.

1.3 Použité pojmy

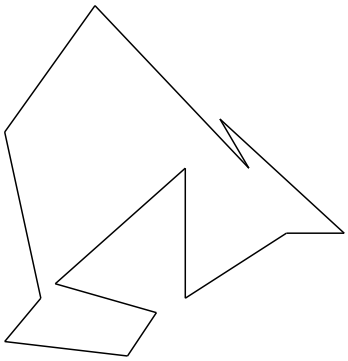
V této části jsou uvedeny pojmy, které jsou využity pro popis algoritmu hledání nejkratší cesty. Některé zde použité anglické termíny nemají přímý český jednoslovný překlad a proto je v textu použito původních anglických termínů, které jsou jednoznačné a lépe srozumitelné. Jedná se o pojem funnel, což v doslovném překladu znamená trychtýř, hourglass což jsou přesýpací hodiny a point locator, což znamená hledač polohy bodů.

Bod, úsečka, vrchol - Bodem je taková uspořádaná dvojice $[x, y]$, kde x a y jsou reálná čísla. Úsečka je taková křivka, která spojuje dva body a to tak, že neexistuje kratší spojení těchto dvou bodů. Vrcholem se pak myslí takový bod, který je krajním bodem úsečky.

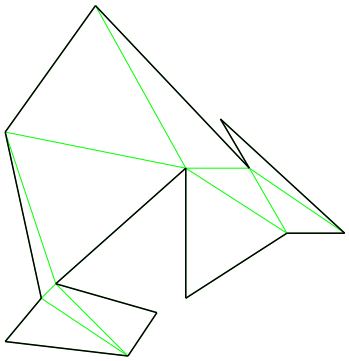
Vzdálenost, délka - Nechť A, B jsou body o souřadnicích $[a_x, a_y]$, $[b_x, b_y]$. Pak se vzdáleností l těchto bodů, které tvoří úsečku, rozumí

$$l = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}.$$

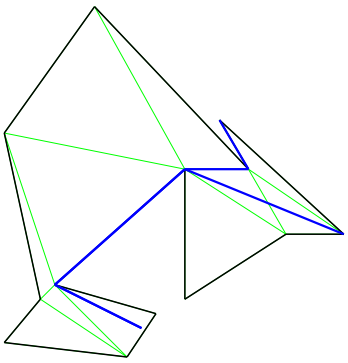
Nechť W je řetězec po sobě jdoucích bodů, kde dva body jdoucí po sobě jsou spojeny úsečkou. Pak součet vzdáleností vrcholů jednotlivých úseček je **délka řetězce** W.



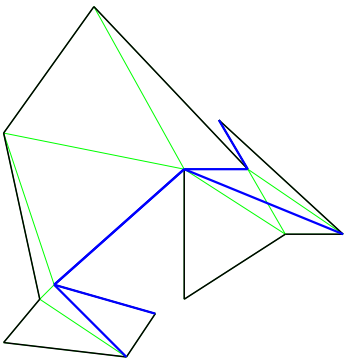
(a) jednoduchý polygon



(b) triangulace jednoduchého polygonu



(c) funnel v jednoduchém polygonu s vyznačenou pomocnou triangulací



(d) hourglass v jednoduchém polygonu

Obrázek 1.1: Příklady vizualizace základních geometrických objektů a struktur.

Jednoduchý polygon - Jednoduchý polygon (viz obr. 1.1a) je takový řetězec po sobě jdoucích úseček, kde dvě po sobě jdoucí úsečky sdílejí své krajní body, poslední úsečka navazuje na první úsečku a tím tvoří cyklus. Úsečky, které po sobě v řetězci nenásledují, spolu žádné body nesdílejí.

Triangulace jednoduchého polygonu - Triangulace jednoduchého polygonu P je takové rozdělení prostoru uvnitř jednoduchého polygonu, kde nově vzniklé oblasti tvoří trojúhelníky, jejichž vrcholy jsou vrcholy P. Příklad triangulace je zobrazen na obrázku 1.1b.

Diagonála - Diagonálou je taková úsečka, která celá leží v polygonu nebo ve volném prostoru.

Funnel v jednoduchém polygonu, Apex - Nechť P je jednoduchý polygon, d zvolená diagonála ležící uvnitř P a A libovolný bod nacházející se uvnitř P. Funnel $f(A, d)$ pak představuje nejkratší spojení z bodu A na vrcholy d. Funnel je tvořen dvěma posloupnostmi bodů značenými F_a a F_b, kde první prvky obou posloupností je bod A, následující

prvky představují vrcholy P a poslední prvky představují vrcholy d. První prvky obou posloupností se mohou shodovat, tj. cesty se částečně překrývají. Poslední prvek, kde se obě posloupnosti shodují se nazývá **apex**. Ten představuje místo, kde se funnel rozděluje a každá posloupnost postupuje dále již samostatně ke svému cíli. Ukázka funnelu je na obrázku 1.1c.

Pro jednoduchý polygon platí, že v něm existuje pouze jedna nejkratší cesta spojující dva body [7]. To znamená, že každá posloupnost funnelu tvoří jedinou možnou posloupnost představující nejkratší cestu. Z toho vyplývá, že každá cesta musí křížit jedinečnou posloupnost diagonál triangulace jednoduchého polygonu, což je předpoklad na kterém je založen algoritmus pro hledání funnelů.

Hourglass jednoduchém polygonu - Hourglass v jednoduchém polygonu představuje všechny nejkratší cesty mezi dvěma zadanými diagonálami. Hourglass je tvořen dvěma posloupnostmi vrcholů, které představují vrcholy jednoduchého polygonu. První prvky a poslední prvky posloupností jsou vrcholy zadaných diagonál. Pokud to je možné, tak se jednotlivé úsečky hourglass neprotínají. Pokud nelze zajistit jejich neprotnutí, tak to znamená, že hourglass v určité části sdílí společné úsečky. Hourglass v jednoduchém polygonu je zobrazen na obrázku 1.1d.

Polygonální doména - Polygonální doména je takový rovinný prostor, který se nachází uvnitř jednoduchého polygonu, označeného jako hraniční, a překážek, které jsou představovány jednoduchými polygony umístěnými uvnitř hraničního polygonu. Prostor mezi hraničním polygonem a překážkami je označen jako volný prostor. Volný prostor zahrnuje i samotné hranice překážek a hraničního polygonu. Ukázka polygonální domény je na obrázku 1.2a.

Mapa - Instance polygonální domény reprezentující pracovní prostředí robotu je v následujícím textu označena jako mapa.

Voroného diagramy - Nechť p je rovina obsahující body množiny $\mathcal{A}$ A_1 až A_n. Voroného diagramy představují rozdělení roviny p do takových oblastí, pro které platí, že každá oblast obsahuje právě jeden bod A_i a všechny body z této oblasti jsou nejblíže z bodů $\mathcal{A}$ právě bodu A_i.

Zobecněný segmentový Voroného diagram pro úsečky a body je rozšíření množiny $\mathcal{A}$ o objekty úseček. Příklad takového Voroného diagramu je na obrázku 1.2b.

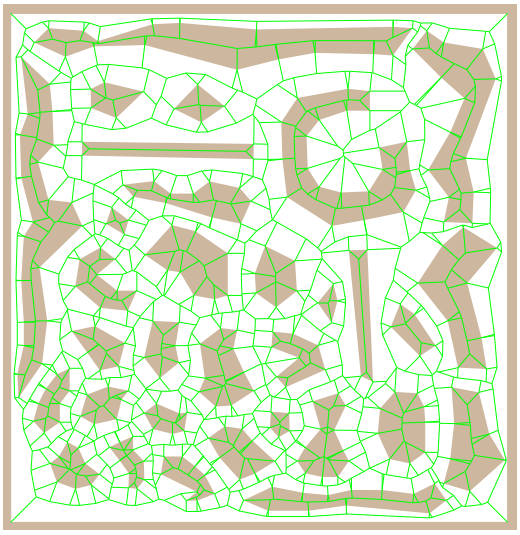
Teselace - Teselace obecně představuje rozdělení volného prostoru na určité oblasti. Algoritmus, který je předmětem této práce využívá takovou teselaci, která rozděluje volný prostoru mapy na trojúhelníky a jednoduché polygony. Tato teselace je založena na segmentovém Voroného diagramu. Příklad zde použité teselace je vidět na obrázku 1.2c.

Oblast - Oblastí je trojúhelník nebo jednoduchý polygon, který vznikly při teselaci.

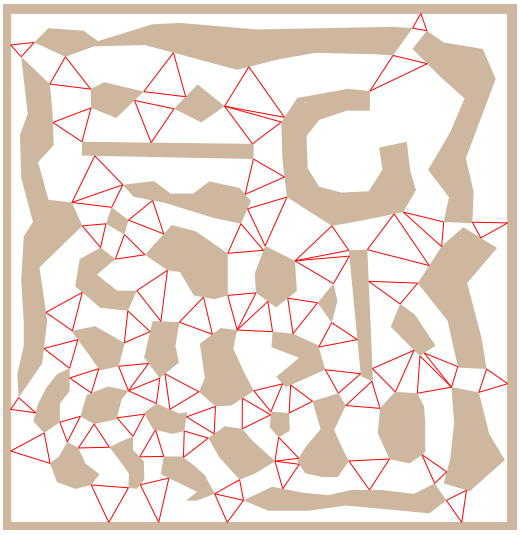
Brána - Brána je taková diagonála, která je společná dvěma oblastem. To znamená, že představuje hranici mezi dvěma oblastmi.

Point locator - Point locator je takový algoritmus, který slouží pro určení oblasti, ve které se nachází zadaný bod. Složitost určení oblasti je $O(\log n)$, kde n je počet vrcholů mapy. Popis algoritmu point locatoru je například v [2].

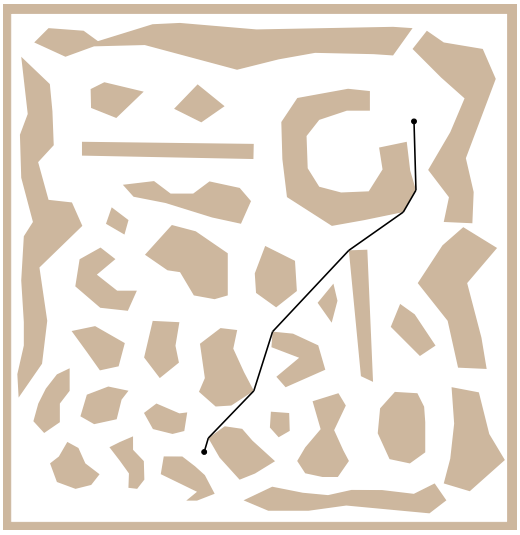
(a) příklad instance polygonální doména



(b) Voroného diagram



(c) příklad teselace polygonální domény



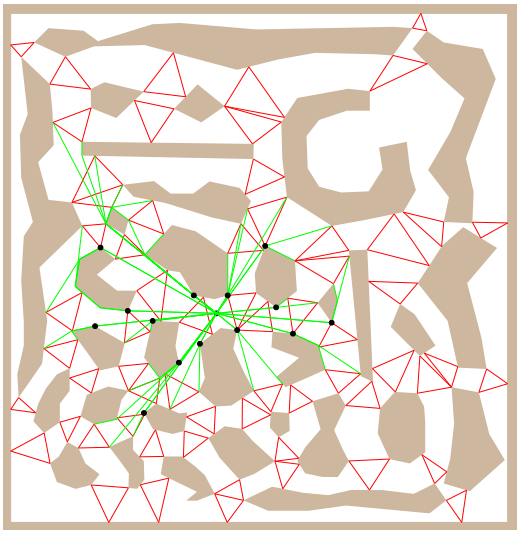
(d) příklad nejkratší cesty

Obrázek 1.2: Příklad polygonální domény, podpůrných struktur a nejkratší cesty.

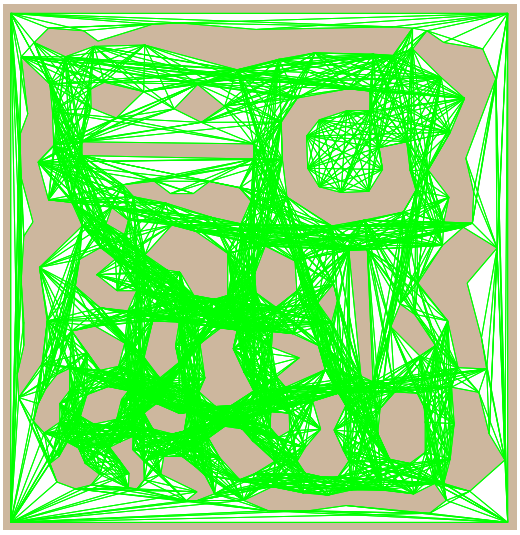
Viditelnost - Dva body A a B jsou vzájemné viditelné právě tehdy, jestliže úsečka, jejíž vrcholy jsou A a B, leží pouze ve volném prostoru mapy.

Nejkratší cesta - Cesta je taková posloupnost bodů, která začíná zdrojovým bodem hledání, následují body nacházející se ve volném prostoru mapy a končí a cílovým bodem hledání. Nejkratší cesta je taková cesta, jejíž délka je minimální mezi všemi cestami. Tato cesta nemusí být jedinečná. Nejkratší cesta musí procházet vrcholy mapy (pokud nejsou hledané body vzájemné viditelné) [2], a každý vrchol cesty je viditelný pouze pro vrchol přímo před ním nebo přímo za ním. Ukázka nejkratší cesty je na obrázku 1.2d

Kritický bod - Pokud zdrojový a cílový bod hledání nejsou vzájemně viditelné, tak existuje takový vrchol mapy, který je viditelný ze zdrojového, respektive cílového bodu a prochází jím nejkratší cesta. Tento bod je označen jako kritický bod. Podrobnosti a důkaz tvrzení lze nalézt v [5].



(a) ukázka všech uzavřených funnelů z jednoho bodu včetně vyznačených možných kritických bodů



(b) úplný graf viditelnosti

Obrázek 1.3: Příklad uzavřených funnles a grafu viditelnosti.

Uzavřený funnel - Uzavřený funnel je takový funnel $f(A, d)$, který má apex různý od zdrojového bodu A. Znamená to, že vytvořený funnel je veden na diagonálu d, která již není přímo viditelná z počátečního bodu funnelu A. Druhý bod jakékolik posloupnosti funnelu představuje možný kritický vrchol. Z důvodu viditelnosti na diagonálu d se otevřené funnely z bodu A nemohou křížit. Ukázka všech uzavřených funnelů z jednoho bodu je na obrázku 1.3a.

Grafy viditelnosti - Graf viditelnosti je graf reprezentující relaci viditelnosti mezi vrcholy mapy, kde vrcholy grafu představují vrcholy mapy a dva vrcholy jsou spojeny hranou právě tehdy, jsou-li odpovídající vrcholy mapy vzájemně viditelné. Cena hrany odpovídá vzdálenosti vrcholů mapy. Grafu viditelnosti je v algoritmu pro hledání nejkratší cesty využito k určení cesty mezi kritickými vrcholy. Ukázka grafu viditelnosti je v vidět na obrázku 1.3b.

Matice vzdáleností jednotlivých vrcholů - Matice vzdáleností je zápis délek jednotlivých cest mezi vrcholy mapy určený prohledáním grafu viditelnosti. Indexy řádků a sloupců odpovídají vrcholům mapy, prvky matice odpovídají délce cesty mezi odpovídajícími vrcholy.

1.4 Poznámky k pseudokódům

V tomto textu je několik algoritmů, které jsou vyjádřeny pomocí pseudokódu. Pro větší čitelnost kódu je zde zavedeno několik konvencí, které jsou v textu algoritmů použity. Body a vrcholy jsou značeny velkými písmeny, například A. Názvy množin jsou značeny kaligrafickými znaky, například $\mathcal{D}$ a výčet jejich prvků je umístěn do složených závorek, $\{X, Y, Z\}$. Posloupnosti jsou značeny velkými písmeny, například F a výčet jejich prvků je umístěn do kulatých závorek, například (X, Y, Z) Pro přídání jednoho prvku

na konec posloupnosti je použit symbol $\oplus$, například $F \oplus X$. Pro označení úsečky je použito páru špičatých závorek, ve kterých se nachází vrcholy úsečky, $\langle A, B \rangle$. Pro označení délky úsečky nebo posloupnosti bodů (vrcholů) je použito vyjádření v absolutních hodnotách, $|F(A, B, C)|$.

Kapitola 2

Algoritmus hledání nejkratší cesty

Úkolem této bakalářské práce bylo implementovat vybraný algoritmus pro hledání nejkratší cesty v polygonální doméně. Byl vybrán algoritmus [5], jelikož představoval nový přístup k problému, který zatím ve skupině Inteligentní robotiky nikdo neimplementoval a navíc se tento algoritmus jevil jako optimální.

Algoritmus popsaný v [5] je založen na hledání kritických bodů a jejich následné využití pro nalezení nejkratší cesty. Nejkratší cesta je nalezena využitím dotazu na mapu nejkratších cest pro kritický bod, kde je hledaná cesta mezi kritickým bodem a bodem zdrojovým, respektive cílovým. Zde popsaný algoritmus není algoritmus popsaný přesně v [5], ale je v něm několik rozdílů. Hlavním rozdílem je využití grafů viditelnosti místo mapy nejkratších cest. Toto vede k potřebě nalézt všechny kritické body pro počáteční i cílový bod, což způsobí zpomalení algoritmu oproti [5]. Toto zjednodušení bylo zavedeno z důvodu malé časové dotace na bakalářskou práci, jelikož implementace mapy nejkratších cest by byla časově náročná.

Algoritmus hledání nejkratších cest inspirovaný dle [5] využívá ke své činnosti několik jiných algoritmů, které jsou v této kapitole uvedeny a popsány. Prvním popsaným algoritmem je postup konstrukce teselace, využité následně pro tzv. expanzi funnelů. Expanze funnelů je pak popsána v samostatné sekci. Po této expanzi dojde k učení možných kritických bodů a z těchto bodů je následně určena výsledná cesta. Výsledkem teselace je rozdělení volného prostoru na trojúhelníky a jednoduché polygony. Pro jednoduché polygony jsou zkonstruovány hourglass spojující brány každého polygonu.

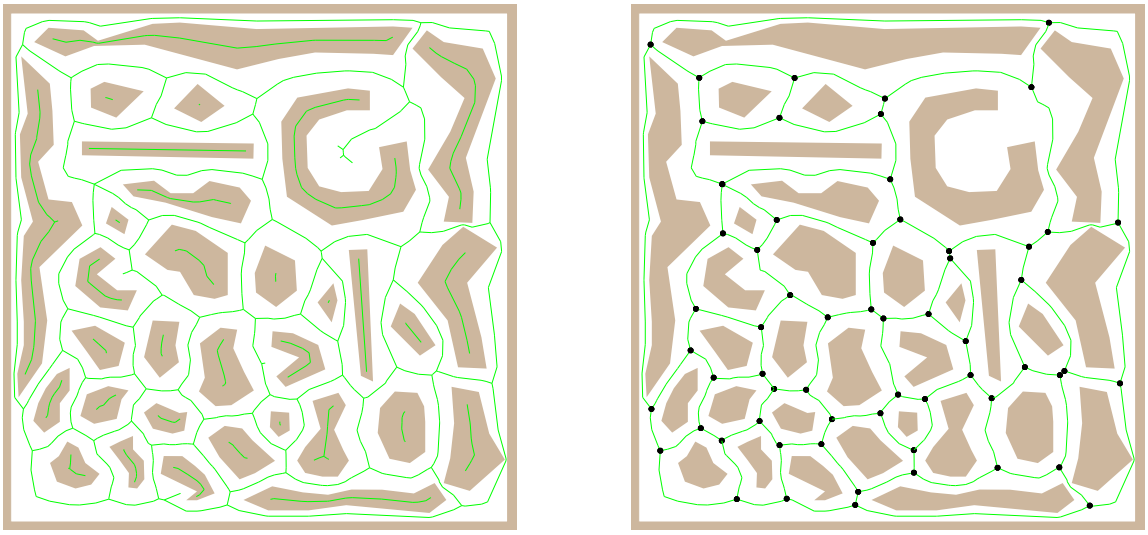
Další skupinou algoritmů jsou algoritmy založené na funnelech v jednoduchém polygonu. Jedná se o konstrukci funnelu, konstrukci hourglass a konstrukci nejkratší cesty v jednoduchém polygonu. Dva poslední algoritmy využívají ke své činnosti funnely zkonstruované algoritmem prvním. Hourglass slouží pro urychlení expanze funnelů v polygonální doméně a algoritmus nejkratší cesty je využit v případě, kdy se oba hledané body nacházejí ve stejné oblasti teselace.

2.1 Algoritmus teselace

Teselace použitá pro algoritmus hledání nejkratší cesty je založena na zobecněném Voroného diagramu. Rozděluje prostor na trojúhelníky a jednoduché polygony takové, které jsou mezi těmito trojúhelníky. Výhodou této teselace je její jednoznačnost a citlivost

na počet překážek. Podmínkou pro algoritmus konstrukce teselace je, že žádné čtyři vrcholy mapy neleží na jedné kružnici, která se celá nalézá ve volném prostoru a jejíž obsah obsahuje pouze volný prostor.

Algoritmus teselace je možné rozdělit na čtyři části, které jsou zde popsány. První část je vytvoření segmentovaného Voroného diagramu. Popis Voroného diagramu je nad rámec rozsahu tohoto textu a je navíc velmi podrobně popsán například v [2]. Další částí při tvorbě teselace je převedení Voroného diagram na redukovanou kostru Voroného diagramu. Toto se provede ve dvou krocích. Prvním krokem je odstranění všech hran Voroného diagramu, které sdílejí jeden svůj vrchol s vrcholem zpracovávané mapy. Ukázka výsledku tohoto kroku je na obrázku 2.1a. Druhým krokem je odstranění všech hran, které mají vrchol stupně jedna a převedení všech hran, které mají vrchol stupně dva na jednu hranu (dvě hrany jsou spojeny vrcholem stupně dva a tyto hrany jsou v redukovaném diagramu vyjádřeny hranou jedinou). Ukázka mapy po tomto kroku je na obrázku 2.1b. Celý tento postup je zapsán symbolicky v algoritmu 1.



(a) (b)

Obrázek 2.1: Tvorba redukované kostry Voroného diagramu: (**a**) Voroného diagram; (**b**) vizualizace druhé části algoritmu konstrukce teselace, kde černé disky představují vrcholy stupně tři.

Třetí částí tvorby teselace je konstrukce trojúhelníků. Po tvorbě redukovaného diagramu je pro každý vrchol stupně tři určen trojúhelník. Pro určení trojúhelníku nastávají tři následující možnosti:

1. **Neexistuje žádný bod třetího stupně** - Tento případ nastane, pokud prohledávaný prostor neobsahuje ani jednu překážku nebo právě jednu překážku. V případě žádné překážky bude použit pouze algoritmus na prohledávaná jednoduchého polygonu. V případě jedné překážky může být volba trojúhelníku libovolná tak, aby jeho vrcholy ležely na úsečkách mapy. Tyto případy nejsou ovšem na přiložené implementaci algoritmu řešeny z časových důvodů jejich implementace.
2. **Bod je incidentní třem různým oblastem** - Pro každou oblast incidentní s V se nalezne nejbližší bod v mapě, který leží na úsečce tvořící mapu. Tyto body se spojí do hledaného trojúhelníku.

Algoritmus 1: Tvorba redukované kostry Voroného diagramu

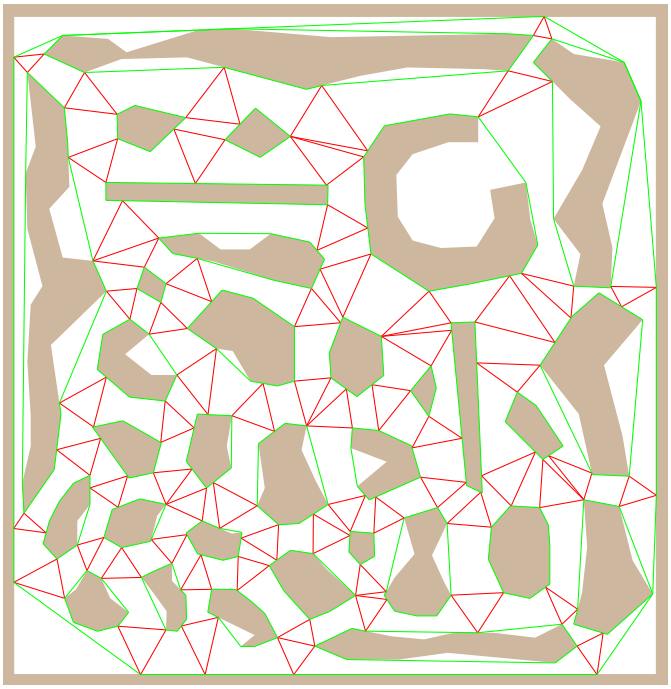
Vstup: $\mathcal{V}$ - Voroného diagram, $\mathcal{V} = G(W, E)$, kde W jsou vrcholy Voroného diagramu a E jsou hrany mezi vrcholy
Vstup: M - Mapa, $M = \{M_1, M_i, ..., M_k\}$, kde M_i jsou vrcholy mapy
Výstup: $\mathcal{V}$ - Redukovaná kostra Voroného diagramu

```
1  foreach V in 𝒲 do
2      if V ∈ M then
3          E := E \ {e, e ∈ E, e je incidentní s V }
4          W := W \ {V}
5  foreach V in W do
6      if V stupně 1 then
7          E := E \ {e, e ∈ E, e je incidentní s V }
8          W := W \ {V}
9  foreach V in W do
10     if V stupně 2 then
11         {V1, V2} := {Vx, Vx ∈ W, Vx spojený hranou s V }
12         E := E \ {e, e ∈ E, e je incidentní s V }
13         E := E ∪ {e, e je hrana mezi V1 a V2 }
14         W := W \ {V}
```

3. **Bod je incidentní dvěma různým oblastem** - Spojí se vrchol V a nejbližší bod leží na úsečce tvořící mapu v samostatné oblasti, označen A. K právě vytvořené úsečce se zkonstruuje kolmice jdoucí bodem V. Nejbližší průsečíky kolmice k bodu V s mapou se spojí spolu s bodem A. Tyto body tvoří trojúhelník.

Čtvrtou a zároveň poslední částí tvorby teselace je identifikace jednoduchých polygonů, které vznikly při tvorbě trojúhelníků a konstrukce hourglass pro každý takto nalezený polygon. Každý jednoduchý polygon sousedí se dvěma trojúhelníky, ovšem trojúhelník může sousedit i s jiným trojúhelníkem. Pro činnost algoritmu expanze funnelů je potřeba ještě identifikovat sousednosti jednotlivých oblastí teselace, jelikož přechodí část algoritmu teselace toho explicitně neprovedla. Tento krok se provede postupným procházením vytvořených trojúhelníků a testováním jejich hran. Pokud je hrana shodná s jiným trojúhelníkem, je pouze určeno, se kterým trojúhelníkem zpracovávaný trojúhelník sousedí. Pokud je testovaná hrana shodná s jednoduchým polygonem, je tento polygon identifikován a opět je určeno s kým sousedí.

Během identifikace jednoduchých polygonů se určí pro každý jednoduchý polygon hourglass. Výpočet hourglass pro jednoduché polygony slouží pro algoritmus prohledávání. Algoritmus prostřednictvím hourglass neprohledává ty vrcholy, kterými nejkratší cesta vést nemůže. Na obrázku 1.1d všechny vrcholy, kterými neprochází zelené úsečky (hourglass) nehrají v prohledávání roli. Pro každý nalezený polygon se provede výpočet hourglass popsaný v kapitole 2.2.3.



Obrázek 2.2: Ukázka všech hourglass.

2.2 Konstrukce funnelu v jednoduchém polygonu

Funnely v jednoduchém polygonu jsou využity pro dva účely. Prvním účelem funnelů v jednoduchém polygonu je nalezení nejkratší cesty mezi dvěma body nacházející se ve stejném polygonu. Při hledání nejkratší cesty je potřeba pak tento případ řešit samostatně právě pomocí funnelů. Druhým účelem je konstrukce hourglass, které slouží na urychlení algoritmu propagace funnelů. Obě tyto konstrukce jsou založeny na funnelech a proto je konstrukce funnelu uvedena v textu jako první. Po popisu konstrukce funnelů následuje popis využití funnelu pro hledání nejkratší cesty v jednoduchém polygonu a v poslední části této sekce je uveden popis konstrukce hourglass s využitím funnelů.

2.2.1 Algoritmus konstrukce funnelů

Zde popsaný algoritmus pro tvorbu funnelů využívá triangulace jednoduchého polygonu. Jelikož každá posloupnost funnelu představuje nejkratší cestu z bodu na vrchol, musí protínat jedinečné pořadí diagonál triangulace. Tyto diagonály mohou být určeny na začátku konstrukce funnelu a pak dále využity pro nalezení funnelu. Algoritmus je popsán ve třech po sobě jdoucích částech: Triangulace, Posloupnost diagonál a Konstrukce výsledného funnelu.

Triangulace První část algoritmu je triangulace prohledávaného polygonu. Popis triangulace není v této práci uveden, jelikož je velmi podrobně popsán například v [2]. V této triangulaci jsou určeny trojúhelníky obsahující počáteční bod funnelu a cílovou diagonálu funnelu. Toto určení je provedeno využitím point locatoru značeného ptLoc($\mathcal{T}$, A), kde $\mathcal{T}$

je prohledávaná triangulace a A je hledaný prvek (bod nebo diagonála). V tomto kroku je možné, že dojde ke zjištění, že hledané trojúhelníky se shodují. V tomto případě se hledaný funnel skládá pouze ze zadaného bodu a vrcholů cílové diagonály a algoritmus konstrukce funnelu končí.

Jsou-li trojúhelníky různé, je mezi těmito trojúhelníky určena posloupnost diagonál, kterou funnel protíná. Na základě této posloupnosti je pak funnel postupně zkonstruován „prodlužováním" funnelu o vrcholy nalezených diagonál.

Určení posloupností diagonál Pro určení posloupností diagonál je potřeba nejprve určit posloupnost trojúhelníků a z této posloupnosti pak určit samotné diagonály. Tato posloupnost následně slouží pro postupnou konstrukci funnelů.

Triangulaci jednoduchého polygonu si lze představit jako graf, jehož vrcholy jsou trojúhelníky, které jsou spojeny hranou, která představuje jejich společnou diagonálu, v kódu je graf značen $G(V, E)$, kde V jsou jeho vrcholy, tedy trojúhelníky a E jsou hrany, tedy diagonály triangulace. V tomto grafu lze pozorovat, že v něm nemohou vzniknout cykly a proto lze použít na jeho prohledání prohledávání do hloubky, které je v kódu značené depthFirstSearch($G(V, E)$, t_A, t_d), kde $G(V, E)$ je prohledávaný graf, t_A je počáteční trojúhelník a t_d je cíloví trojúhelník. Toto prohledání určí posloupnosti trojúhelníků nacházející se mezi trojúhelníkem obsahující zdrojový bod hledání a trojúhelníkem obsahující cílovou diagonálu hledání.

Z nalezené posloupnosti trojúhelníků se postupuje od počátku k cíli. Do výsledné posloupnosti diagonál se vždy přidá diagonála (hranu grafu), kterou se cesta přesouvá do dalšího trojúhelníku. Tento postup je formálně zapsán v algoritmu 2.

Algoritmus 2: FindDiagonalSequence

Vstup: P - Jednoduchý polygon, ve kterém je funnel konstruován
Vstup: A - Počáteční bod funnelu, $A \in P$
Vstup: d - Cílová diagonála funnelu, $d \in P$
Výstup: D - Posloupnost diagonál, které výsledný funnel protíná

1 $\mathcal{T}$:= triangulate(P), $\mathcal{T} = G(V, E)$
2 t_A := ptLoc($\mathcal{T}$, A)
3 t_d := ptLoc($\mathcal{T}$, d)
4 **if** $t_A = t_d$ **then**
5 Algoritmus konstrukce funnelu končí a vrací informaci, že body jsou vzájemně viditelné.
6 T := depthFirstSearch($G(V, E)$, t_A, t_d), $T = (t_1, ..., t_{n-1}, t_n)$
7 **for** $t_i := t_1$ **to** t_{n-1} **do**
8 $D := D \oplus d_x, d_x \in E$, d_x je incidentní t_i a t_{i+1}

Určení samotného funnelu Konstrukce výsledného funnelu v jednoduchém polygonu se provádí pomocí „prodlužování" funnelu vrcholy nalezených diagonál a následném zjednodušení zkonstruovaného funnelu. Tento postup je uveden v algoritmu 3, na jehož začátku je vytvořen tzv. inicializační funnel, který je zobrazen na obrázku 2.3. Tento funnel začíná v zadaném počátečním bodě a jeho posloupnosti končí na vrcholech první diagonály D.

Po vytvoření tohoto funnelu přejde algoritmus do hlavní smyčky, ve které postupně funnel rozšiřuje. V této smyčce jsou postupně zpracovány všechny diagonály v posloupnosti diagonál. Toto rozšíření je provedeno tak, aby výsledný funnel byl co nejkratší. Toho je dosaženo tak, že je nalezen vrchol s nejnižším indexem v rozšířené posloupnosti, který lze spojit s rozšiřujícím bodem takovou úsečkou, která celá leží v zadaném polygonu. Po tomto rozšíření se testuje, zda neexistuje kratší cesta do přidaného bodu přes posloupnost funnelu, která nebyla rozšířena. Toto se provádí opět přes hledání nejnižšího indexu vrcholu v posloupnosti nerozšířené takového, že úsečka spojující tento vrchol a bod přidaný celá leží v polygonu. Pokud je takto nalezená cesta kratší než cesta rozšířené posloupnosti, je rozšířená posloupnost nahrazena. Postup tvorby funnelu je uveden na obrázku 2.4.

Algoritmus 3: ConstructFunnel

Vstup: P - Jednoduchý polygon
Vstup: A - Zdrojový bod hledání
Vstup: D - Posloupnost diagonál, která výsledný funnel protíná. Posloupnost je ve tvaru $D = (d_0, ..., d_i, ..., d_n)$, kde $d_i = (K_i, L_i)$, kde K_i, L_i jsou vrcholy d_i a d_n je cílová diagonála.
Výstup: F_a, F_b - Posloupnosti funnelu

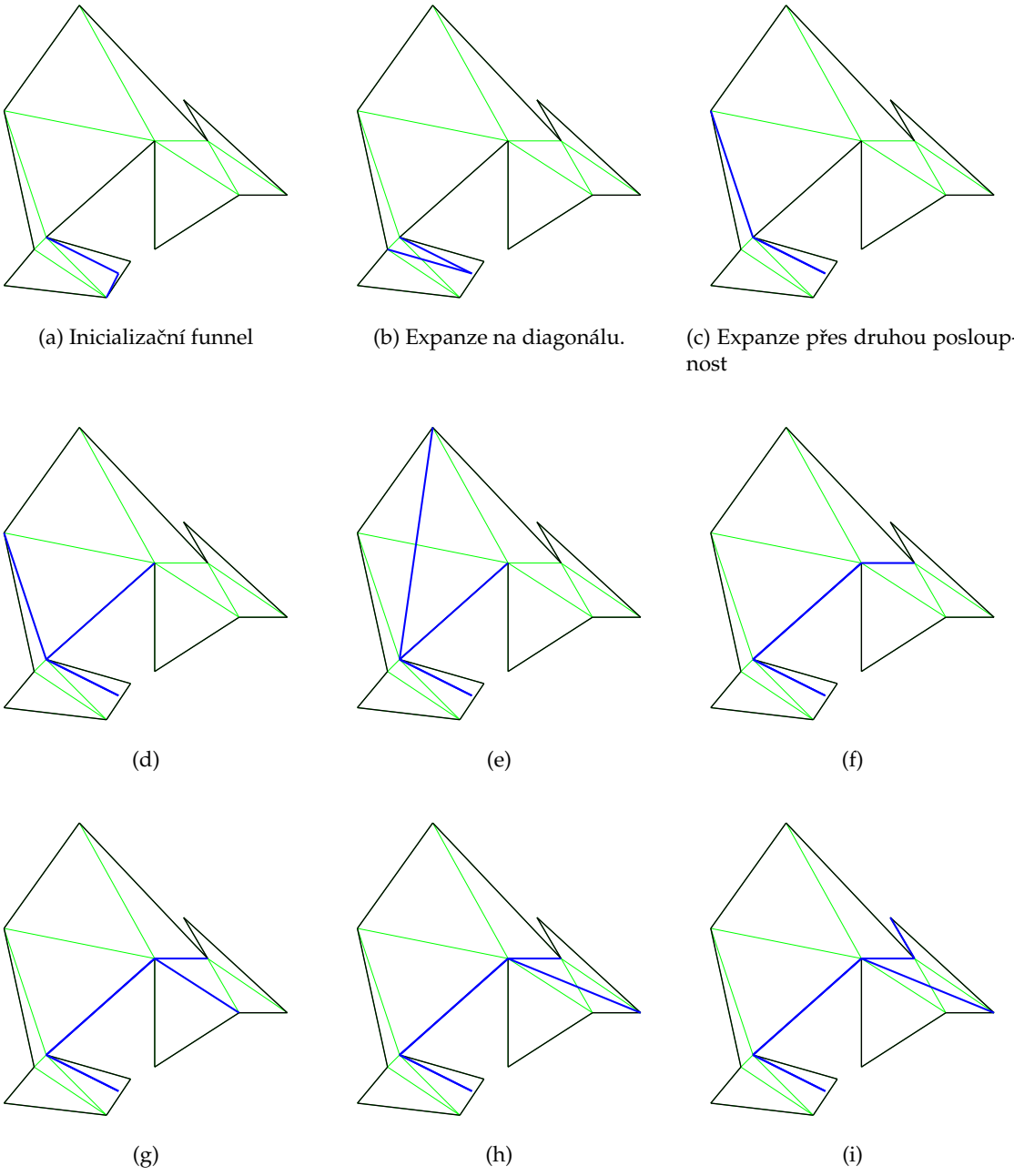
1 $F_a := (K_0)$
2 $F_b := (L_0)$
3 **for** $d_i := d_1$ **to** d_n **do**
4 $\quad F_x := F$, F není incidentní s $d \wedge F \in \{F_a, F_b\}$, $F_x = (X_1, ..., X_k)$
5 $\quad F_y := \{F_a, F_b\} \setminus F_x$, $F_y = (Y_1, ..., Y_l)$
6 $\quad M := V$, $V \in \{L_i, K_i\} \wedge$ není incidentní s F_y
7 $\quad i := min_{i=1,..,k}\langle X_i, M\rangle \in P, X_i \in F_x$
8 $\quad F_x := (X_1, ..., X_i, M)$
9 $\quad j := min_{i=2,..,l}\langle Y_i, M\rangle \in P, Y_i \in F_y$
10 $\quad$ **if** $|Y_1, ..., Y_j, M| < |F_x|$ **then**
11 $\quad\quad F_x := (Y_1, ..., Y_j, M)$

2.2.2 Algoritmus hledání nejkratších cest v jednoduchém polygonu

Algoritmus pro hledání nejkratších cesty v jednoduchém polygonu je založen na funnelech. Postup je velmi podobný konstrukci funnelu na diagonálu. Pouze s tím rozdílem, že do posloupnosti diagonál se přidá na konec taková diagonála, která má jeden svůj vrchol incidentní s vrcholem prohledávaného polygonu a druhý vrchol má incidentní s cílovým bodem cesty. Výsledná cesta je pak ta posloupnost funnelu, která končí hledaným bodem. Pseudokód je zobrazen na 4.

2.2.3 Konstrukce hourglass

Jelikož hourglass představuje množinu cest mezi dvěma diagonály a funnel představuje množinu cest mezi bodem a diagonálou, je možno využít funnelů pro konstrukci hourglass. Tento algoritmus konstruuje hourglass vyžitím dvou funnelů vedoucích z vrcholů jedné diagonály na diagonálu druhou. Každý funnel se skládá ze dvou posloupností a činností této části algoritmu je vybrat dvě posloupnosti ze čtyř posloupností funnelů, které tvoří hourglass mezi zadanými diagonály. Tento výběr probíhá na základě



(a) Inicializační funnel

(b) Expanze na diagonálu.

(c) Expanze přes druhou posloupnost

(d)

(e)

(f)

(g)

(h)

(i)

Obrázek 2.3: Postup tvorby funnelu, jednotlivé posloupnosti jsou vyznačeny modře.

Algoritmus 4: FindShortestPathInSimplePolygon

Vstup: P - jednoduchý polygon
Vstup: A - Zdrojový vrchol $A \in P$
Vstup: B - Cílový vrchol $B \in P$
Výstup: F - Posloupnost nejkratší cesty

```
1  T := triangulation(P)
2  t_A := ptLoc(T, A)
3  t_B := ptLoc(T, B)
4  D := FindDiagonalSequence(T, t_1, t_2)
5  d_2 := ⟨B, libovolný vrchol t_2⟩
6  D := D ⊕ d_2
7  f(A, d_2) := ConstructFunnel(P, A, D), f(A, d_2) = {F_a, F_b}, F_a = (A_1, ..., A_k)
8  if A_k = B then
9  |  F:= F_a
10 else
11 |  F:= F_b
```

délek jednotlivých posloupností a na základě protnutí jednotlivých posloupností. Prvním testem je, zda se kratší posloupnosti obou funnelů nekříží. Pokud ne, jsou tyto posloupnosti hledaným hourglass. Pokud se kříží, tak se testuje, zda nemají společnou úsečku. Pokud mají, tak jsou tyto posloupnosti hledaným hourglass a znamená to, že v určitém místě je hledaný hourglass degenerovaný pouze na úsečku. Pokud ani tato podmínka není splněna, jsou hledaným hourglass delší posloupnosti funnelů. Detail postupu je vidět v algoritmu 5.

2.3 Algoritmus hledání nejkratší cesty v polygonální doméně

V této části popisu je již připravena teselace a v ní předpočítané hourglass v jednoduchých polygonech. Tato teselace je využita a její využití je popsáno v této sekci. Dále jsou v této části využity grafy viditelnosti mezi vrcholy mapy, které je potřeba předpočítat a z nich určit matici vzdáleností pro jednotlivé vrcholy mapy.

Algoritmus je založen na hledání kritických bodů, mezi kterými nalezne cestu využitím predpočítaného grafu viditelnosti mezi vrcholy (pokud nejsou dotazované body vzájemně viditelné). Tento přístup je správný, jelikož nejkratší cesta vede přes vrcholy mapy mezi kterými lze najít cestu grafem viditelnosti. Kritické vrcholy se hledají využitím propagace funnelů.

Algoritmus je možné rozdělit do několika dílčích kroků, které se postupně provedou a jsou zde popsány v tomto pořadí. Prvním krokem je nalezení kritických bodů pomocným algoritmem expanze funnelů z obou krajních bodů cesty. Po nalezení množiny možných kritických bodů jsou z této množiny vybrány dva body, pro které platí, že cesta jimi vedoucí je nejkratší možná. Výběr těchto bodů se provede dotazem matice vzdáleností s přičtením vzdáleností zkoumaných kritických bodů a krajních bodů cesty. Po nalezení těchto bodů je prohledán graf viditelnosti vrcholů mapy a tím je nalezena cesta mezi kritickými body a tím je i nalezena cesta mezi zadanými body, jelikož zadané body a kritické body jsou vzájemně viditelné.

Algoritmus 5: ConstructHouglass(P, d_1, d_2)

Vstup: P - jednoduchý polygon
Vstup: d_1 - Zdrojová diagonála $d_1 \in P$, vrcholy V_1, W_1
Vstup: d_2 - Cílová diagonála $d_2 \in P$, vrcholy V_2, W_2
Výstup: H - Hledaný hourglass

```
1  T := triangulate(P)
2  D := FindDiagonalSequence(T, V1, d2)
3  f(V1, d2) := ConstructFunnel(P, V1, D)
4  f(V1, d2) = {Fs, Fr}
5  D := FindDiagonalSequence(T, V2, d2)
6  f(V2, d2) := ConstructFunnel(P, V2, D)
7  f(V2, d2) = {Ft, Fu}
8  Fa := F, F ∈ {Fs, Fr}, |F| = min(|Fs|, |Fr|)
9  Fb := F, F ∈ {Ft, Fu}, |F| = min(|Ft|, |Fu|)
10 Fc := F, F ∈ {Fs, Fr}, |F| = max(|Fs|, |Fr|)
11 Fd := F, F ∈ {Ft, Fu}, |F| = max(|Ft|, |Fu|)
12 if Fa nemá společný bod s Fb then
13 |   H := (Fa, Fb)
14 else
15 |   if Fa má společnou úsečku s Fb then
16 |   |   H := (Fa, Fb)
17 |   else
18 |   |   H := (Fc, Fd)
```

2.3.1 Algoritmus hledání možných kritických bodů

Vstupem této části algoritmu je mapa s připravenou teselací a výstupem je pak množina možných kritických bodů. Hlavní část algoritmu je založena na propagaci funnelů, tj. jejich „prodlužování" směrem od zdrojového bodu dál. Každý funnel je propagován tak dlouho, dokud není uzavřen. Pokud jsou všechny funnely uzavřeny, algoritmus propagace končí. Pak se z vytvořených funnelů vezmou vždy druhé body z jejich posloupností. Z jaké posloupnosti funnelu jsou body převzaty nehraje roli, jelikož jsou oba body stejné. Tyto body představují množinu možných kritických bodů. Algoritmus končí i v případě, kdy zjistí, že dotazované body jsou vzájemně viditelné. Postup hledání možných kritických bodů je popsán ve čtyřech hlavních částech.

První částí je identifikace oblastí teselace, ve které leží zdrojový a cílový bod cesty. Pro tento účel je opět využit point locator. Po tomto kroku se porovná, zda jsou oblasti $R(a)$ a $R(b)$ stejné. Pokud jsou a oblasti jsou trojúhelníky, algoritmus vrátí informaci, že jsou body vzájemně viditelné. Pokud jsou stejné a jedná se o jednoduchý polygon, tak se použije algoritmus pro hledání nejkratší cesty v jednoduchém polygonu a tuto cestu si zapamatuje a po dokončení expanze porovná délku cesty uvnitř polygonu a vně polygonu a vrátí cestu kratší. Tato varianta může nastat, jelikož oblasti teselace nemusí být konvexní polygony a tudíž nalezená cesta v rámci jednoduchého polygonu nemusí představovat nejkratší cestu mezi zadanými body mapy. Funkce realizující tuto část je uvedena v algoritmu 7.

Po identifikaci oblastí jsou zkonstruovány tzv. inicializační funnely. Tyto funnely ve-

Algoritmus 6: LocateRegions($\mathcal{M}$, A, B)

Vstup: $\mathcal{M}$ - Mapa s teselací připravená algoritmem
Vstup: A - Zdrojový bod $A \in M$
Vstup: B - Cílový bod $B \in M$
Výstup: t_1, t_2 - Trojúhelníky obsahující zdrojový bod, cílovou diagonálu
Výstup: F - Možná nejkratší cesta v rámci jednoduchého polygonu

```
1 t1 := ptLoc(M, A)
2 t2 := ptLoc(M, B)
3 if t1 = t2 then
4 |   F :=FindShortestPathInSimplePolygon(P, A, B)
5 else
6 |   F := ∅
```

dou od zdrojového (respektive cílového) bodu na brány oblasti, ve které se bod nachází, Pro každou bránu oblasti se zkonstruuje funnel ze zadaného bodu na bránu této oblasti. Každý takto vytvořený funnel se ověří, zda není uzavřen. Pokud není, je přidán do množiny otevřených funnelů, pokud je uzavřen, je přidán do množiny uzavřených funnelů. Funkci popsanou uvedenou v algoritmu **??** je nutné zavolat na obě zadané oblasti, tj. na oblast počáteční i na oblast cílovou. V této funkci je využita triangulace právě zpracovávané oblasti. Tuto triangulace je možno převzít z části algoritmu teselace, kde byly vytvářeny hourglass.

Po konstrukci inicializačních funnelů a jejich uložení do správné množiny (otevřených nebo uzavřených funnelů) je provedena hlavní část algoritmu a to expanze jednotlivých funnelů. Vstupem pro tuto část jsou oblasti obsahující zdrojový, respektive cílový bod cesty, dále značené $R(a)$ a $R(b)$. Dalším vstupem je množina otevřených, respektive uzavřených funnelů pro počáteční a cílový bod cesty, značené $F_O(a)$ a $F_O(b)$, což jsou otevřené funnely a $F_C(a)$ a $F_C(b)$, což jsou uzavřené funnely. Výstupem této části algoritmu jsou množiny uzavřených funnelů, které přišly na vstup rozšířené o všechny nalezené a uzavřené funnely. Postup algoritmu expanze je následující:

1. Z množiny otevřených funnelů $F_O(a)$ se postupně vybírají jednotlivé funnely $f(A_1, d)$ dokud není množina prázdná. Vybraný funnel se z množiny odstraní. Na vybraný funnel je aplikován krok 2. Je-li prázdná, pokračuje se algoritmem v části o výběru kritických bodů.

2. $f(A_1, d)$ se expanduje na oblast R, která je s d incidentní a zároveň jí $f(A_1, d)$ ještě neprochází.

3. Pokud je R shodná s oblastí $R(b)$, tak se zkonstruuje funnel z bodu B na bránu, přes kterou bylo expandováno a provede se spojení tohoto funnelu s funnelem $f(A_1, d)$. Následně se provede zjednodušení dle bodu 6.

4. Pokud je oblast R trojúhelník, tak dojde k rozdělení $f(A_1, d)$ na dva nové funnely. Každý nový funnel pokračuje na bránu, kterou žádný funnel ještě neprochází. Nové funnely se vytvoří tak, že na poslední místa jejich posloupností se přidají okrajové body diagonály, na kterou expandují a to tak, aby se výsledné úsečky funnelu nekřížily.

Algoritmus 7: ConstructInitialFunnels

Vstup: R - Oblast teselace
Vstup: A - Bod $A \in R$
Výstup: $F_O(A)$ - Množina otevřených funnelů začínajících v bodě A
Výstup: $F_C(A)$ - Množina uzavřených funnelů začínajících v bodě A

```
1  if R je trojúhelník then
2      g1, g2, g3 := brány(R)
3      gi = (G1i, G2i), gi ∈ {g1, g2, g3}
4      f(A, g1) := {(A, G11), (A, G21)}
5      FO(A) := FO(A) ∪ {f(A, g1)}
6      f(A, g2) := {(A, G12), (A, G22)}
7      FO(A) := FO(A) ∪ {f(A, g2)}
8      f(A, g3) := {(A, G13), (A, G23)}
9      FO(A) := FO(A) ∪ {f(A, g3)}
10 else
11     T := triangulate(R)
12     g1, g2 := brány(R)
13     t1,t2 := LocateTriangles(T, A, g1)
14     D := FindDiagonalSequence(T, t1, t2)
15     f(A, g1) := ConstructFunnel(R, A, D)
16     if uzavřen(f(A, g1)) then
17         FO(A) := FO(A) ∪ {f(A, g1)}
18     else
19         FC(A) := FC(A) ∪ {f(A, g1)}
20     t1,t2 := LocateTriangles(T, A, g2)
21     D := FindDiagonalSequence(T, t1, t2)
22     f(A, g2) := ConstructFunnel(R, A, D)
23     if uzavřen(f(A, g2)) then
24         FO(A) := FO(A) ∪ {f(A, g2)}
25     else
26         FC(A) := FC(A) ∪ {f(A, g2)}
```

5. Pokud je oblast R jednoduchý polygon, tak dojde rozšíření $f(A_1, d)$ o hourglass polygonu R. To znamená, že do posloupností $f(A_1, d)$ se přidají posloupnosti z hourglass a to tak, aby se výsledné úsečky funnelu nekřížily.

6. Tento krok provede zjednodušení nově vzniklých nebo upravených funnelů. Je popsán pouze pro jeden funnel s posloupnostmi $F(a)$ a $F(b)$, kde $F(a)$ je posloupnost, která bude zjednodušena. Je potřeba jej zavolat pro obě posloupnosti funnelu. V případě expanze trojúhelníku je potřeba tento krok provést pro oba vzniklé funnely a všechny jejich posloupnosti.

   (a) Z $F(a)$ se postupně vybírají body (značeny P_e) od posledního bodu posloupnosti do posledního bodu, který byl přidán z hourglass v případě expanze na jednoduchý polygon. V případě expanze na trojúhelník je vybrán pouze bod poslední. V případě, že byl zpracován poslední bod, přejde se na krok 1.

   (b) Z $F(a)$ se postupně vybírají body (P) od posledního apexu $F(a)$ do posled-

ního bodu $F(a)$, který byl přidán z hourglass v případě expanze na jednoduchý polygon, v případě expanze na trojúhelník k předposlednímu bodu $F(a)$. V případě, že byl zpracován poslední bod, přejde se na krok 6e.

(c) Spojí se úsečkou k bod P_e a bod P.

(d) V případě, kdy nedojde k průniku k a úseček sestavených z posloupností $F(a)$ a $F(b)$ a zároveň k leží v polygonu sestaveném z úseček $F(a)$ a $F(b)$, odmažou se z posloupnosti $F(a)$ všechny body mezi P_e a P, a pokračuje se na krok následující. V případě průniku se přejde na krok 6b.

(e) Postupně se projde druhá posloupnost funnelu $F(b)$ a vyberou se z ní body P od posledního apexu k poslednímu bodu $F(b)$. Vybrané body P se ukládají do pomocné posloupnosti $F(c)$. V případě, že byl zpracován poslední bod, přejde se na krok 1.

(f) Spojí se úsečkou k bod P_e a bod P.

(g) V případě, kdy nedojde k průniku k a úseček sestavených z posloupností $F(a)$ a $F(b)$ a zároveň k leží v polygonu sestaveném z úseček $F(a)$ a $F(b)$, tak se do posloupnosti $F(c)$ přidají body zpracované z kroku 6a. Porovná se délka posloupností $F(b)$ a $F(c)$. V případě, že délka $F(c)$ bude menší než $F(b)$, tak se nahradí posloupnost $F(b)$ za posloupnost $F(c)$ a právě zpracovávaný funnel se přesune do množiny $F_C(a)$. Algoritmus přejde na krok 1. V případě průniku se přejde na krok 6e.

Po nalezení všech uzavřených funnelů se algoritmus dostane do části, kdy ze všech uzavřených funnelů určí kritické body. Jelikož je uzavřený funnel takový funnel, který má apex různý od počátečního bodu, je možným kritickým bodem každého uzavřeného funnelu jeho apex. A jelikož byl funnel uzavřen, jakmile se stal jeho apexem jakýkoliv jiný bod, než jeho počátek, tak je kritickým bodem první bod (přesněji vrchol) následující po bodu počátečním. Tato část algoritmu je popsána pouze pro $F_C(a)$. Pro $F_C(b)$ se použije stejný algoritmus, pouze s jiným funnelem. Postupně se z množiny $F_C(a)$ vybírají funnely $f(A, d)$. Z funnelu se vezme první vrchol nejbližší bodu A. Tento bod se uloží do množiny možných kritických bodů C_A.

2.3.2 Algoritmus hledání nejkratší cesty

Toto je poslední část algoritmu hledání nejkratší cesty, ve které je již určena výsledná cesta. Vstupem této části jsou zdrojový a cílový bod hledání (dále značené jako C_A C_B), množina kritických bodů nalezených v předchozí části a dále se zde pak vyskytuje matice vzdáleností mezi vrcholy mapy a graf viditelnosti mezi vrcholy mapy. Tento algoritmus jako první určí, které body jsou kritické. Toto určení provede pomocí matice vzdáleností.

Postupně se projdou všechny vrcholy v C_A (dále značeny A_A) a určí se vzdálenost A_A a zdrojového bodu, označena je dále l_1. Pro každý zpracovávaný vrchol z C_A se postupně projdou všechny vrcholy v C_B (dále značeny A_B) a určí vzdálenost A_B a cílového bodu, označena je dále l_2. Pak se dotáže matice vzdáleností na vzdálenost vrcholů A_A a A_B a k této vzdálenosti přičte l_1 a l_2. Z těchto vypočtených vzdáleností se najde vzdálenost minimální a pro tu platí, že vrcholy P_A a P_B jsou kritické. Tento krok hledání nejkratší cesty je uveden v algoritmu 8.

Dále algoritmus provede prohledání grafu viditelnosti pro nalezení nejkratší cesty mezi vrcholy P_A a P_B. Toto prohledání je možné například pomocí Floyd-Warshallova

algoritmu. Takto nalezená cesta je nejkratší cestou mezi zadanými body. Na začátek této cesty se přidá zdrojový bod hledání a na její konec se přidá cílový bod hledání. Takto sestavená cesta představuje nejkratší možnou cestu (přesto tyto cesty mohou existovat i dvě různé, ale obě stejně dlouhé) mezi dvěma zadanými body mapy.

Algoritmus 8: ShortestPathQuery(A, B, C_A, C_B)

Vstup: A - Zdrojový bod
Vstup: B - Cílový bod
Vstup: C_A - Množina možných kritických bodů pro zdrojový bod
Vstup: C_B - Množina možných kritických bodů pro cílový bod
Výstup: P - Nejkratší cesta mezi A a B

1 $i, j := min_{i=1,\ldots,n;j=1,\ldots,m}(|C_{Ai}A|$ + distmatrix(C_{Ai}, C_{Bj}) + $|C_{Bj}B|)$
2 $P := (A, C_{Ai}, visGraph(C_{Ai}, C_{Bj}), C_{Bj}, B)$

Kapitola 3

Experimenty

Nad implementovaným algoritmem bylo provedeno několik experimentů za účelem zjištění a ověření jeho vlastností. Testy byly provedeny za účelem kontroly implementace, tj. korektnosti výsledků, otestování časové náročnosti implementace. Výsledky testů byly porovnány s řešením nalezeným algoritmem grafů viditelnosti a aproximačním řešením.

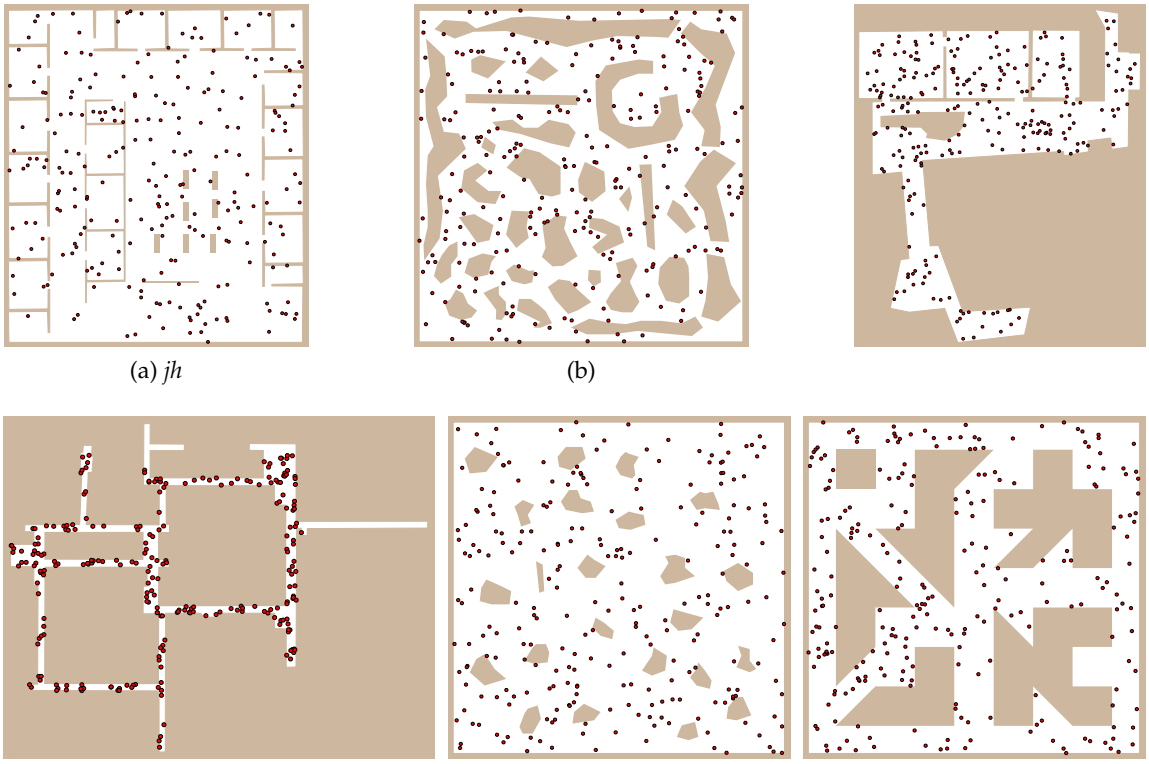
Prvním testem bylo vložení 250 náhodných bodů do map jh, dense, ta, pb, potholes a m2. Mezi všemi 250 body byla hledána cesta od každého ke každému, tj. 62 250 dotazů na nejkratší cestu celkem. Vizualizace testovaných bodů je vidět na obrázku 3.2 a ukázka vybraných cest na obrázku **??**. V tomto testu byla měřena vzdálenost, která byla určená z nalezené cesty a zaznamenávám čas jednotlivých dotazů. Cesta byla vždy měřena v obou směrech a to pro porovnání konzistence výsledků. Na základě změřené vzdálenosti bylo provedeno porovnání s referenčním řešením získaného pomocí grafu viditelnosti a dále provedeno porovnání s aproximačním řešením.

3.1 Test správnosti a náročnosti

V testu správnosti algoritmu porovnáním s řešením grafů viditelnosti byly výsledky následující: Mapy jh, dense, ta, potholes a m2 vracely stejné výsledky jako reference, mapa bp vracela 490 chyb, kde průměrná chyba nepřesáhla 1 bod. Při vizualizaci chyb nebyly nelezeny žádné nedostatky cest a tak pravděpodobně došlo k zaokrouhlovací chybě v typu double.

V tomto testu byl zároveň hodnocen čas jednotlivých dotazů a paměťová náročnost podpůrných datových struktur. Průměrné časy pro jednotlivé mapy jsou vidět v tabulce 3.2 a histogramy časů pro každou mapu na obrázcích 3.3. V tabulce 3.2 jsou vidět počty vrcholů map N_V, množství překážek v mapě N_H, počet trojúhelníků teselace N_T, počet jednoduchých polygonů teselace N_P, inicializační čas T_{init}, paměťová náročnost přípravy C_{init} a čas trvání průměrného dotazu T_{query}. Na jednotlivých histogramech je vidět, jak jsou konktrétní mapy pro program náročné. Čím více je viditelných překážek z počátečního, respektive cílového bodu, tím je výsledný čas větší. To proto, že při expanzi projdou funnely více trojúhelníkama, jejichž celkový počet odpovídá celkovému počtu překážek, a tím pádem je jich daleko více. Toto odpovídá tvrzení o algoritmu, že jeho složitost je závislá na počtu překážek.

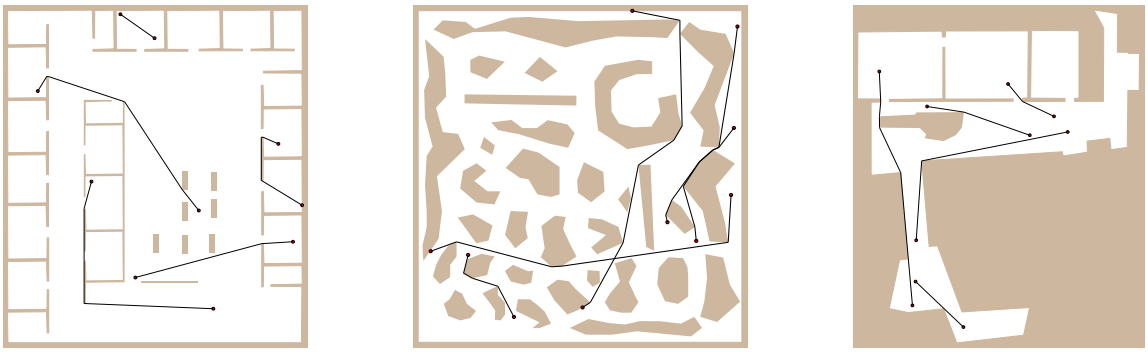
Při profilování implementovaného algoritmu byly výsledky takové, že velká většina času se stráví právě při expanzu funnelů. Jedním z možných příčin tohoto výsledku je vy-



(a) *jh* (b)

(d) (e) (f)

Obrázek 3.1: Ukázka jednotlivých map v pořadí jh, (b) dense, ta, pb, potholes, m2.



(a) (b) (c)

Obrázek 3.2: Ukázka řešení několika cest v mapách jh, dense, ta.

užití implementace postavené na knihovně CGAL, kde bylo nutné použít exaktní kernel (což je základní datový typ kterým jsou reprezentovány základní geometrické objekty), protože u jiného kernelu byl program velmi nespolehlivý a často docházelo k jeho předčasnému ukončení. Této komplikace by se šlo vyhnout využitím jiného přístupu pro výpočet základních geometrických operacích.

U implementovaného algoritmu je vidět velký rozdíl v časech přípravy datových struktur a průměrných časů dotazů. Tento rozdíl je potřeba při praktickém využití algoritmu brát v potaz a snažit se maximálně využít již předpočítané pomocné struktury, tj. snažit se maximalizovat počet dotazů na jednu připravenou mapu.

Tabulka 3.1: Tabulka průměrných časů implementovaného exaktního algoritmu

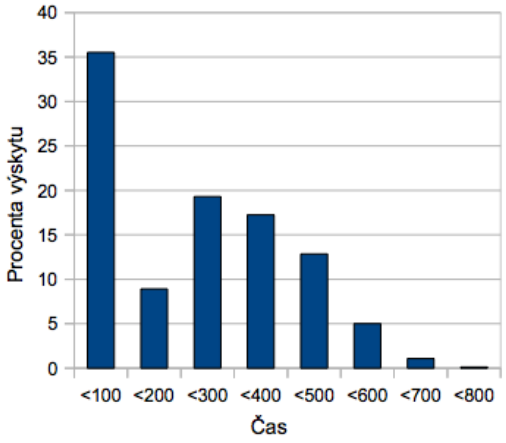
Mapa	N_V	N_H	N_T	N_P	T_{init} [ms]	C_{init} [kb]	T_{query} [ms]
jh	196	9	16	22	1567,00	2 737	226,53
dense	288	32	62	91	2936,00	5 144	28,34
ta	74	2	2	3	506,00	772	97,78
bp	89	3	4	6	678,00	856	68,34
potholes	153	23	44	66	838,00	2 172	47,03
m2	51	6	10	15	373,00	611	9,80
Úplný průměr					1149,67	2 049	79,64

3.2 Porovnání algoritmů pro hledání nejkratších cest

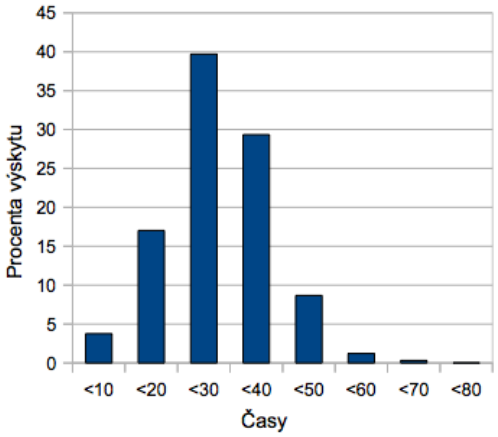
V této části textu jsou porovnány jednotlivé exaktní a aproximačni algoritmy. U exaktních algoritmů jsou porovnávány časy jednotlivých dotazů a časy příprav podpůrných datových struktur. U algoritmů aproximačních jsou jednak porovnány časy a jednak jsou porovnány přesnosti (kvality) jednotlivých cest. V tabulce **??** je uvedeno porovnání aproximačního algoritmu t_{aprox} [4], algoritmu vistor t_{visrot}, což je algoritmus konstrukce grafu viditelnosti, který je popsán v [8] a algoritmu t_{visll}, který je opět variantou konstrukce grafu viditelnosti a je popsán například v [2].

Aproximace zde porovnávaná využívá pro svoji činnost teselaci, která rozdělí volný prostor na konvexní polygony. V těchto polygonech je zajištěno, že jakékoliv dva body se v něm nacházející je možné spojit přímo. Tohoto je využito a hledání nejkratší cesty probíhá konstrukcí úseček, jejichž vrcholy jsou na hranících jednotlivých polygonů. Poté se algoritmus snaží opět pomocí testovacích úseček nalezenou cestu zjednoduššit (princip podobný zjednoduššení funnelu).

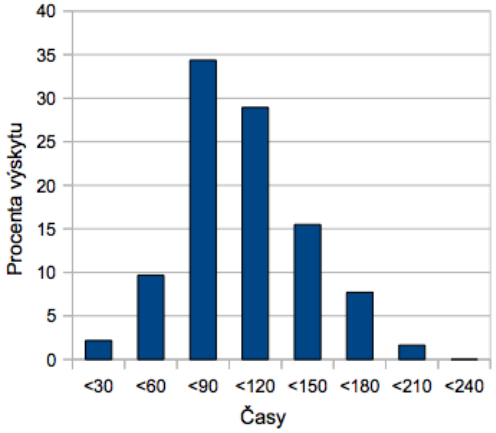
V tabulce 3.3 jsou uvedeny přesnosti aproximace porovnané s referenčním řešením. Byla porovnánáva dvě kritéria a to průměrná chyba délky nejkratší cesty a množství cest, které aproximace určila souhlasně s referenčním řešením. Aproximační řešení byla provedena dvě a to s větší a menší hrubostí aproximace. Průměrné výsledky jsou výjádřeny v procentech vždy vztažených k počtu řešení (L_e) a počet správných řešení je vyjádřen v procentech vztažených k celkovému počtu řešení (n_c). Z tabulky je patrné, že aproximace vrací nejlepší výsledky na mapě potholes. Je to z toho důvodu, že velké množství



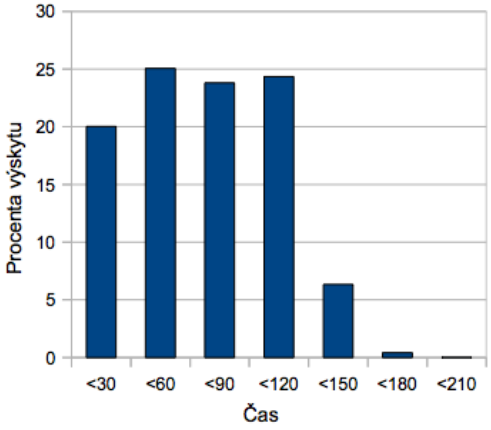
(a) jh



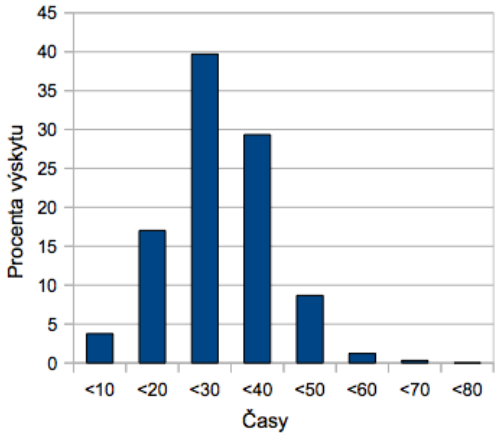
(b) dense



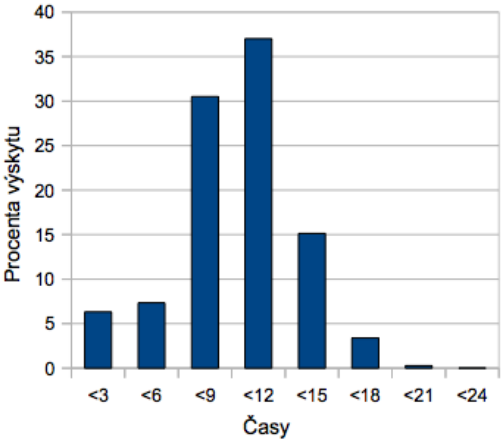
(c) ta



(d) pb



(e) potholes



(f) m2

Obrázek 3.3: Histogramy časů dotazů pro testované mapy.

Tabulka 3.2: Tabulka průměrných časů exaktních algoritmu a aproximace.

Mapa	t_{aprox} [μs]	t_{visrot} [ms]	t_{visll} [ms]	$t_{implementace}$ [ms]
jh	5,07	1,51	18,17	226,53
dense	1,06	3,02	37,10	28,34
ta	1,75	0,33	7,02	97,78
bp	2,12	0,40	9,26	68,34
potholes	0,55	1,25	12,10	47,03
m2	0,53	0,20	4,43	9,90
Úplný průměr	1,85	1,12	14,68	79,64

testovacích bodů na této mapě je vzájemně viditelných, což je aproximace schopna odhalit. Nejhorší výsledek naopak dává mapa jh, kde je umístěna velká překážka přes velikost mapy, se kterou měla aproximace největší problémy.

Tabulka 3.3: Tabulka průměrných chyb aproximací.

Mapa	**Hrubá approximace**		**Jemná approximace**	
	L_e%	n_c%	L_e%	n_c%
jh	17,15	17,30	12,79	17,15
dense	10,68	7,09	10,68	7,09
ta	14,16	11,45	14,16	11,45
bp	16,37	10,93	16,37	10,93
potholes	0,69	33,11	0,69	0,69
m2	11,73	17,65	11,73	17,65
Průměr	11,80	16,26	11,07	16,23

3.3 Řešení úlohy obchodního cestujícího

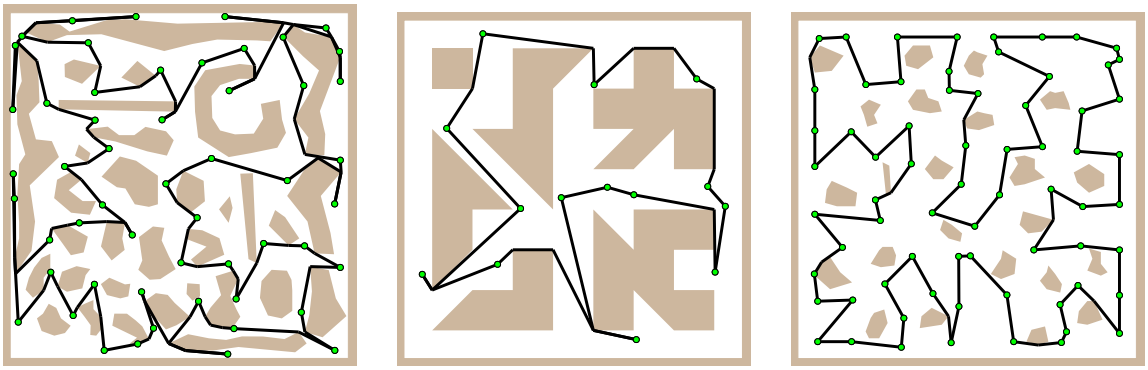
Problémem obchodního cestujícího je najít na mapě takovou cestu, která spojuje zadaná města mapy, která jsou reprezentována body v mapě. Algoritmus se snaží o minimalizaci délky této cesty. Úloha obchodního cestujícího dle [4] je řešena pomocí samo organizující se neuronové sítě. Tato síť pracuje tak, že vytvoří pro každé město samostatný neuron. Všechny neurony jsou uspořádány v určité posloupnosti jak jdou za sebou. Na začátku jsou neurony vloženy do mapy a postupně jsou vybírány tak, že k náhodně vybranému městu se najde nejbližší neuron. Tento neuron je posunut o určitou vzdálenost po nejkratší cestě mezi neuronem a městem. Neurony, které jsou „sousedi" v posloupnosti jsou posunuty také o určitou vzálenost k vybranému městu, ovšem menší, než neuron vybraný. Tento proces se opakuje do té doby, než se najde řešení obchodního cestujícího s délkou vyhovující zadané podmínce.

Tento experiment má určit, zda využití exaktního přístupu k výpočtu vzdálenosti

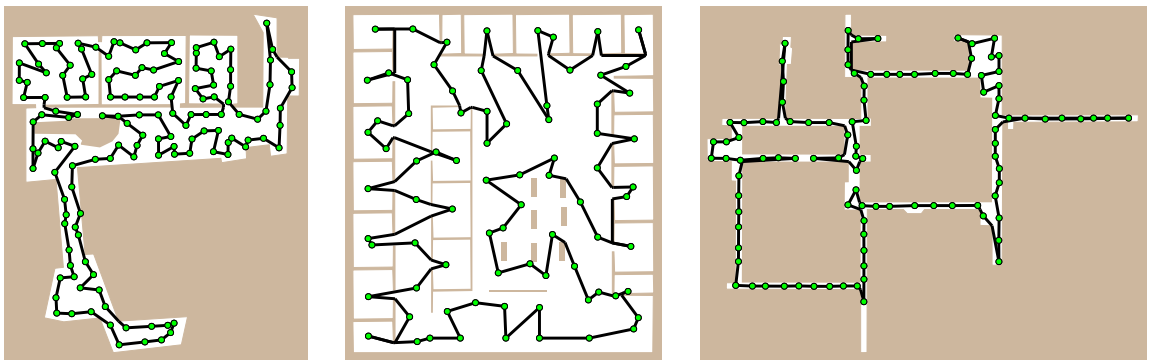
mezi městem a neuronem dosáhne lepších výsledků než aproximace zde uvedená. V tabulce 3.4 jsou uvedeny výsledky tohoto experimentu, kde n je počet měst v mapě, L_{ref} je referenční délka řešení, PDM je procenty vyjádřený roptyl délky cesty vzhledem k referenční délce, PDB je procenty vyjádřený rozpyl nejkratší délky cesty vzhledem k referenčnímu řešení , s_L je roptyl všech délek cest a T je čas řešení v sekundách.

Tabulka 3.4: Porovnání řešení TSP s exaktní a přibližnou nejkratší cestou

Problém	n	L_{ref} [m]	**Nejkratší cesta**				**Approximace dle [4]**			
			PDM	**PDB**	$s_L\%$	T [s]	**PDM**	**PDB**	$s_L\%$	T [s]
jari	6	13,6	0,31	0,00	0,76	0,12	0,00	0,00	0,00	0,01
complex2	8	58,5	0,29	0,00	1,31	0,14	0,00	0,00	0,00	0,01
m1	13	17,1	1,07	0,00	1,92	0,71	0,03	0,00	0,05	0,02
m2	14	19,4	10,99	5,32	2,97	0,66	7,73	0,00	4,18	0,02
map	17	26,5	4,43	0,00	3,77	1,29	3,02	0,00	2,64	0,04
potholes	17	88,5	2,88	0,00	2,35	4,11	1,97	0,00	2,64	0,09
a	22	52,7	0,39	0,00	0,45	3,14	0,65	0,00	1,65	0,09
rooms	22	165,9	0,92	0,00	0,66	2,39	0,83	0,00	0,70	0,06
dense_4	53	179,1	16,63	8,30	3,73	59,92	14,65	6,90	4,39	0,62
potholes_2	68	154,5	5,81	3,14	1,43	134,77	5,60	3,82	1,05	0,55
jh_2	80	201,9	2,20	0,43	0,97	241,75	2,01	0,63	0,90	0,78
pb_4	104	654,6	1,05	0,03	1,98	225,41	1,23	0,18	1,89	1,02
ta_2	141	328,0	3,40	2,02	0,75	1 429,74	3,30	1,67	0,90	1,80
potholes_1	282	277,3	7,41	5,53	0,98	17 176,42	7,14	5,01	1,22	8,10
$\text{pb}_{1.5}$	415	839,6	2,96	1,48	2,54	26 066,12	2,62	1,35	2,09	15,57
h2_5	568	1 316,2	2,80	2,00	0,51	129 952,41	2,93	1,91	0,66	66,90
ta_1	574	541,1	6,26	5,05	0,51	139 799,47	6,12	4,82	0,72	30,68



(a) dense$_4$ (b) m2 (c) potholes$_2$



(d) ta (e) jh (f) pb

Obrázek 3.4: Ukázky řešení úlohy obchodního cestujícího.

Kapitola 4

Závěr

Cílem této práce bylo seznámit se s úlohou hledání nejkratších cest v polygonální doméně. Proto byly nejdříve nastudovány přístupy založené na grafu viditelnosti a dále pak přístupy využívající urychlujících struktur a to jak SSQ algoritmy tak i APQ algoritmy. Kromě exaktních algoritmů byl také nastudován algoritmus aproximační. Na základě zjištěných informací o možnostech řešení tohoto problému byl pro implementaci vybrán algoritmus [5], zejména pro jeho citlivost na počet překážek a také proto, že ve skupině inteligentní a mobilní robotiky toto řešení dosud nikdo neimplementoval.

Popis algoritmu uvedený v [5] je velmi stručný a tak bylo nutné určité kroky domyslet. Mezi těmito kroky byla teselace, která byla v původním textu popsána jen velmi hrubě a dále pak způsob dotazů na SPM, který v textu explicitně uveden nebyl. Autoři se v textu odkazují na plnou verzi článku, kterou se však nepodařilo dohledat. Bohužel se ani nepodařilo úspěšně kontaktovat autoři článku. Po detailním nastudování principu algoritmu bylo přistoupen k implementaci. Vzhledem k tomu, že určité části algoritmu jsou velmi náročné na implementaci, která není dostupná, a k časové dotaci na řešení bakalářské práce bylo přistoupeno k několika zjednodušením. Zejména bylo nahrazeno využití mapy nejkratších cest grafem viditelnosti mezi vrcholy map, což vede na výpočetně náročnější část determinování finální cesty z množiny kritických míst. Dále byla v implementaci použita knihovna CGAL [1] pro základní geometrické operace a algoritmy point-locator, triangulace a konstrukce Voroného diagramu. Nevýhodou této knihovny je skutečná výpočetní náročnost algoritmů, která zejména důsledkem využití exaktních datových typů, jež jsou pro použité algoritmy nezbytné. To se projevilo na celkovém času dotazu na nejkratší cestu, jak je vidět ve výsledcích v předchozí kapitole.

Správnost implementace byla experimentálně ověřena v řadě problémů, pro které algoritmus vracel shodné výsledky jako implementace založená na grafu viditelnosti. Výpočetní náročnost implementovaného algoritmu dle [5] je výrazně vyšší než u ostatních testovaných přístupů, jak je patrné z experimentálních výsledků. Důvodem může být implementace podpůrných algoritmů pracujících pouze s datovou reprezentací `double` a zvýšená výpočetní složitost v důsledku nahrazení SPM grafem viditelnosti. Při použití SPM není nutné determinovat všechny možnosti propojení kritických bodů jako konců hledané cesty, ale stačí nalézt pouze možné kritické body od jednoho vrcholu.

V závěru práce byl exaktní přístup hledání cest využit v algoritmu samoorganizujících se sítí pro úlohu obchodního cestujícího v polygonální doméně. Z výsledků vyplývá, že využití exaktního algoritmu nevede na kvalitnější řešení, navíc je časová náročnost výrazně vyšší než při použití aproximačního algoritmu hledání nejkratší cesty.

Literatura

[1] CGAL, Computational Geometry Algorithms Library. Http://www.cgal.org.

[2] de Berg, M.; van Kreveld, M.; Schwarzkopf, O.: *Computational Geometry*. Sprigner, druhé vydání, 2000.

[3] Chen, D.; Daescu, O.; Klenk, K.: *On geometric path query problems*. 1125 Coloner By Drive, Ottawa, Canada: Springer, Heidelberg, 1997.

[4] Faigl, J.; Kulich, M.; Vonásek, V.; aj.: An Application of Self-Organizing Map in the non-Euclidean Traveling Salesman Problem. *Neurocomputing*, ročník 74, 2011: s. 671–679, doi:10.1016/j.neucom.2010.08.026.

[5] Guo, H.; Maheshwari, A.; Sack, J.: *Computational Geometry*. 1125 Coloner By Drive, Ottawa, Canada: School of Computer Science, Carleton University, 2008.

[6] J.Hershberger; Suri, S.: *An optimal algorithm for Euclidean shortest paths in the plane*. Siam J. Computing, 1999.

[7] L.Gubias; J.Hershberger: *Optimal shortest path quiries in a simple polygon*. Computer Science Department, Standford University, 1989.

[8] Overmars, M. H.; Welzl, E.: New methods for computing visibility graphs. V *SCG '88: Proceedings of the fourth annual symposium on Computational geometry*, New York, NY, USA: ACM, 1988, s. 164–171.

[9] Rosen, C.: Shakey the Robot. 1972, http://www.sri.com.

Obsah CD

Přiložené CD obsahuje zdrojové kódy algoritmu hledaní nejkratší cesty a text bakalářské práce ve formátu PDF a zdrojové kódy celého textu pro systém LaTeX. V následující tabulce je popsána struktura CD.

Adresář	Popis
`src`	zdrojové kódy algoritmu
`doc`	zdrojové kódy textu bakalářské práce
`thesis.pdf`	text bakalářské práce

Tabulka 1: Adresářová struktura na CD