Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Efektívne vyhľadávanie ciest v real-time stratégiách

(diplomová práca)

Peter Gatial

Odbor: Informatika 9.2.1
Vedúci: RNDr. Michal Forišek, PhD.

Bratislava, 2010

Abstrakt

Gatial Peter, Efektívne vyhľadávanie ciest v real-time stratégiách. Diplomová práca, Katedra Informatiky, Fakulta Matematiky, Fyziky a Informatiky, Univerzita Komenského, Bratislava, vedúci diplomovej práce: RNDr. Michal Forišek PhD., Bratislava 2010.

Vyhľadávanie ciest je častý problém v IT oblasti. Je to rozhodujúca časť každej real-time stratégie, pretože je to práve vyhľadávanie ciest čo robí hry real-time. Hry so slabým alebo príliš pomalým vyhľadávaním nie sú hrateľné.

Táto práca vysvetľuje ako fungujú najznámejšie a najrozšírenejšie algoritmy. Prináša zlepšenie dolnych ohraničení nového algoritmu PCD, definuje novú waypoint heuristiku a navrhuje modifikáciu algoritmu Landmark A*. Zaoberá sa tiež ich praktickým použitím a časťou tejto práce je aj aplikácia vizualizujúca spomínané algoritmy.

Kľúčové slová: Pathfindig, AStar, Dijkstra, BFS, PCD, Landmark, Multithreaded.

Abstract

Gatial Peter, Effective pathfinding in real-time strategies. Master thesis, Department of Computer Science, Faculty of Mathematics, Physics and Informatics, Comenius University, Bratislava, thesis advisor: RNDr. Michal Forišek PhD, Bratislava 2010.

Pathfinding is common problem in the computer science. It is crucial part of any real-time strategy game as it is what makes game real-time. Games with poor or slow pathfinding are not playable.

This paper explains how the most famous and widely used pathfinding algorithms work. It brings imrovment of lower bounds of new PCD algorithm, defines new waypoint heuristic and proposes modification of Landmark A* algorithm. It also deals with their practical usage and a part of this paper is also an application visualizing all mentioned algorithms.

Key words: Pathfindig, AStar, Dijkstra, BFS, PCD, Landmark, Multithreaded.

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím citovaných zdrojov.

. . . . . .

Ďakujem vedúcemu mojej diplomovej práce RNDr. Michalovi Forišekovi PhD. za pomoc a cenné námety pri písaní tejto práce.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Kapitola 1

Úvod

Real-time stratégie (RTS) dnes patria k najpopulárnejším počítačovým hrám a veľmi dôležitou súčasťou týchto hier je vyhľadávanie ciest na mape. Keďže ide o real-time hry, vyhľadávanie musí prebiehať veľmi rýchlo, pretože hra musí okamžite reagovať na pokyny hráča. Nemôžeme si dovoliť kvôli vyhľadávaniu zastaviť hru čo i len na pár sekúnd - hľadanie musí prebiehať rádovo v milisekundách.

Keďže súčasťou RTS je aj grafika a umelá inteligencia, tak nemôžeme celú dostupnú pamäť venovať len problému hľadania ciest. Je známy Dijkstrov algoritmus, ktorý okrem samotného grafu nepotrebuje pre hľadanie ciest žiadne ďalšie informácie, ale v praxi je na veľkých grafoch kvôli jeho časovej zložitosti nepoužiteľný. Opačným príkladom je predpočítanie a uloženie vzdialeností všetkých dvojíc vrcholov. Tento prístup umožňuje hľadanie vzdialeností vrcholov v konštantnom čase, ale vyžaduje až $\mathcal{O}(n^2)$ pamäte, čo je už pri malých grafoch neprípustné.

Preto treba nájsť efektívny spôsob ako rýchlo hľadať cesty s použitím najviac $\mathcal{O}(n)$ pamäte. Pri vyhľadávaní si môžeme pomôcť predspracovaním mapy (grafu) a uložením si niektorých dôležitých informácii (routovanie medzi regiónmi, predpočítať a zapamätať si skutočnú vzdialenosť do okolitých políčok, atď.), ktoré využijeme pre rýchle hľadanie ciest. Jediným obmedzením je, aby množstvo týchto informácií bolo lineárne od veľkosti grafu. V tejto práci preskúmame niekoľko známych grafových algoritmov, implementujeme ich, navrhneme ich zlepšenia, navrhneme novú heuristiku a algoritmy porovnáme.

Prvým prínosom tejto práce je zlepšenie dolných ohraničení pre vzdialenosť $d(s, t)$ pri hľadaní ciest v algoritme Precomputed Clusters Distances

(PCD). S použitím o $\mathcal{O}(n)$ viac pämate prehľadá v priemere o 10%-13% vrcholov menej a čas behu je o 6%-11% kratší ako pôvodný algoritmus.

Ďalším prínosom je nová waypoint heuristika, ktorá počíta dolné a horné ohraničenia podobným spôsobom ako algoritmus PCD. Keďže táto heuristika nie je konzistentná, tak nemôže byť efektívne použitá v algoritme A* pre hľadanie najkratších ciest. Napriek tomu ohraničenia vypočítané touto heuristikou sú porovnateľné s ohraničeniami v PCD a sú využívané v algoritme Landmark A* Bounded na zredukovanie počtu prehľadávaných vrcholov.

Posledným prínosom je algoritmus Landmark A* Bounded, ktorý je modifikáciou existujúceho algoritmu Landmark A*. Odlišuje sa od neho použitím PCD a waypoint ohraničení na zredukovanie počtu prehľadávaných vrcholov. Rovnako ako PCD využíva fakt, že ak je známe horné ohraničenie $\hat{d}(s,t)$ pre najkratšiu cestu z vrchola s do vrchola t, tak susedné vrcholy ľubovoľného vrchola w nemusia byť prehľadané, ak dolné ohraničenie $\underline{d}(s,w,t)$ ľubovoľnej cesty z s do t cez w je väčšie ako $\hat{d}(s,t)$. Vtedy totiž s určitosťou vieme, že w neleží na najkratšej ceste medzi vrcholmi s a t.

1.1 Štruktúra diplomovej práce

- Kapitola 2 oboznamuje čitateľa so základnými pojmami, vysvetľuje úlohu vyhľadávacích algoritmov v RTS a zavádza niektoré definície, ktoré budeme v práci používať.
- Kapitola 3 popisuje najznámejšie algoritmy riešiace danú problematiku. Detailnejšie je popísaný algoritmus A* a vplyv heuristickej funkcie na jeho správanie.
- Kapitola 4 predstavuje dva nové algoritmy, PCD a Landmark A*, ktoré sa objavili v posledných rokoch a výrazne prekonávajú predchádzajúce algoritmy. Ďalej je tu definovaná nová waypoint heuristika, zlepšenie dolných ohraničení v PCD a nový algoritmus Landmark A* Bounded.
- Kapitola 5 obsahuje výsledky praktických testov porovnávajúcich algoritmy. Je tu popísaný spôsob generovania máp a testov, rozdelenie grafu na bloky a výber waypoint-ov. Nakoniec tu sú zanalyzované získané výsledky z testovania a na ich základe sú sformulované odporúčania pre efektívne vyhľadávania ciest v RTS.
- Kapitola 6 sumarizuje prácu a jej hlavné výsledky.

Kapitola 2

Základné pojmy

V tejto kapitole si definujeme niekoľko základných pojmov, na ktoré sa budeme neskôr odvolávať. V krátkosti čitateľa oboznámime s konceptom RTS a dôležitosťou hľadania optimálnych ciest. Definujeme si pojem hracej mapy a k nej prislúchajúci graf, na ktorom už budeme riešiť problémy využitím poznatkov z teórie grafov.

2.1 Real-time stratégie

Od vydania prvej RTS Dune II už uplynulo niekoľko rokov, ale základné princípy a pravidlá zostali nezmenené. V hre súperia dvaja a viac hráčov (hráč môže byť aj počítač), pričom sa pokúšajú navzájom eliminovať alebo dosiahnuť iný cieľ ako napríklad zničiť určitý počet nepriateľských jednotiek alebo získať určité množstvo suroviny v hre. K tomu slúžia rôzne hracie jednotky, ktoré sa môžu po hernej mape presúvať. Optimálny pohyb jednotiek po mape je teda pre víťazstvo kľúčový.

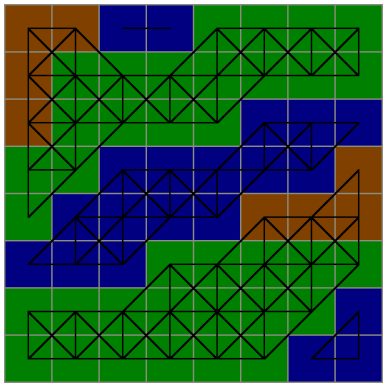
RTS však nemusia byť len bojové, ale aj staviteľské ako napríklad Transport Tycoon. V tejto hre je cieľom stavať cesty a železnice a prepravovať po nich rôzne suroviny v čo najkratšom čase. Počítačový protivník teda potrebuje vedieť nájsť optimálnu cestu medzi dvomi miestami. Problém môže byť ešte náročnejší, ak uvažujeme aj možné úpravy terénu pri stavbe týchto ciest. V týchto hrách by nemusela byť časová zložitosť použitých algoritmov tak dôležitá ako v bojových RTS, keďže počítač môže najlepšie riešenie hľadať priebežne dlhšiu dobu. Zrejme by ani nebolo pre hru hráča vhodné, ak by počítač vždy vedel takmer okamžite postaviť perfektnú cestu.

2.2 Reprezentácia hernej mapy

V tejto práci sa budeme venovať hľadaniu najkratších ciest v 2D mapách. V RTS hrách sú mapy tvorené rôznymi typmi terénu, ktoré ovplyvňujú rýchlosť pohybu jednotiek po nich. Zvyčajne tiež nie je dovolený prechod medzi všetkými typmi terénu, čim v mapách vznikajú prirodzené prekážky a priamočiare smerovanie k cieľu nie je možné. Preto pod pojmom mapa budeme v tejto práci rozumieť hernú mapu popísanú v nasledujúcej definícii.

Definícia 2.2.1 Herná mapa je dvojrozmerné pole políčok NxN, kde $N = 2^k,\ k \geq 1$. Každé políčko má pridelený typ terénu z množiny T = {voda, tráva, hlina, skaly}. Dovolený je prechod medzi všetkými dvojicami typov terénov, okrem dvojíc (voda, tráva), (voda, hlina) a (voda, skaly).

K hernej mape zostrojíme neorientovaný graf $G = (V, E)$ nasledovne. Vrcholmi grafu G budú jednotlivé políčka mapy. Dva vrcholy budú spojené hranou, ak sú susedné v hracej mape a je dovolený prechod medzi ich typmi terénu. Všetky hrany budú ohodnotené kladne. Príklad mapy a jej grafu je na obrázku 2.1, kde modré políčka predstavujú vodu, zelené trávu, hnedé hlinu a čierne úsečky sú hrany medzi vrcholmi.



Obr. 2.1: Mapa a k nej prislúchajúci graf

2.3 Heuristika

Všetky algoritmy spomínané v tejto práci vo všeobecnosti pracujú nasledovne. Udržujú si množinu zatvorených vrcholov $\mathcal{Z}$ a množinu otvorených

vrcholov $\mathcal{O}$. Zatvorené vrcholy sú tie, do ktorých už poznáme cestu zo štartového vrcholu a otvorené sú práve prehľadávané vrcholy. Na začiatku algoritmu je v $\mathcal{O}$ len štartovací vrchol. Algoritmus v každom kroku vyberie podľa nejakého kritéria najlepší vrchol u z množiny otvorených a presunie ho medzi zatvorené. Preskúma všetkých neuzavretých susedov vrchola u, vypočíta ich vzdialenosť od štartového vrchola prechádzajúc cez u a presunie ich do množiny otvorených vrcholov.

Dijkstrov algoritmus vyberá vrchol s najmenšou vzdialenosťou od štartového políčka. Efektívnejšie algoritmy využívajú heuristickú funkciu $h(n)$, definovanú pre každý vrchol n v grafe, ktorá vypočíta odhadovanú vzdialenosť z vrchola n do cieľa t. Výber vrchola potom závisí od hodnoty tejto funkcie. BeFS pri výbere vrchola z množiny otvorených uvažuje len hodnotu $h(n)$, pri A* je to súčet $g(n) + h(n)$, kde $g(n)$ je dĺžka cesty zo štartového vrcholu do vrchola n.

Definícia 2.3.1 Heuristika h je **prípustná**, ak pre každý vrchol n, $0 \leq h(n) \leq h^*(n)$, kde $h^*(n)$ je skutočná dĺžka cesty z vrchola n do cieľa. Inými slovami odhadovaná dĺžka je vždy menšia alebo rovnaká ako skutočná dĺžka cesty do cieľa.

Definícia 2.3.2 Heuristika h je **konzistentná**, ak pre každú dvojicu vrcholov u a v, $h(u) \leq d(u,v) + h(v)$, kde $d(u,v)$ je dĺžka optimálnej cesty medzi u a v. To znamená, že hodnota $h(v)$ nie je menšia než $h(u)$ o viac ako $d(u,v)$.

2.4 Formulácia problému

Majme hernú mapu a k nej zostrojený graf $G = (V, E)$, $|V| = n$, $|E| = m$. Nech $s, t \in V$ a nech typ terénu políčok prislúchajúcich k s a t je z množiny T' = {tráva, hlina, skaly}. Úlohou je pre vrcholy s a t nájsť najkratšiu cestu, pričom môžeme využívať ľubovoľné predpočítané informácie o grafe, ale ich pamäťová zložitosť môže byť najviac $\mathcal{O}(n)$.

Kapitola 3

Vyhľadávacie algoritmy

V tejto kapitole sa pozrieme na niekoľko najznámejších a najpoužívanejších algoritmov na hľadanie ciest. Ako sa ukáže, prvé dva budú špeciálnym prípadom posledného, ktorému sa budeme venovať podrobnejšie.

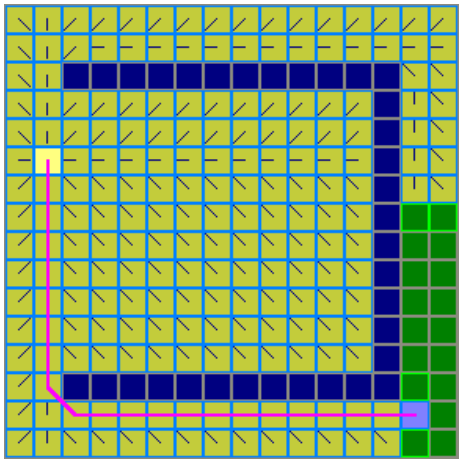
3.1 Dijkstrov algoritmus

Dijkstrov algoritmus, pomenovaný podľa svojho objaviteľa Edsgera Dijkstra, patrí medzi najznámejšie grafové algoritmy. Tento algoritmus rieši problém nájdenia najkratšej cesty v grafe s nezáporne ohodnotenými hranami v čase $\mathcal{O}(n^2)$. Existuje však efektívnejšia implementácia pomocou Fibonacciho haldy s časovou zložitosťou $\mathcal{O}(m + n \log n)$ [MLF87].

Formálne, nech $G = (V, E)$ je graf s nezáporne ohodnotenými hranami. Algoritmus pre daný vrchol $s \in V$ nájde najkratšiu cestu z s do v pre všetky $v \in V$. Algoritmus začína vo vrchole s a postupne prechádza cez všetky vrcholy grafu. V každom cykle vyberie z nenavštívených vrcholov najbližší k s a zistí vzdialenosť všetkých susedov tohoto vrchola od vrchola s. Tento postup opakuje pokiaľ sú v grafe nenavštívené vrcholy, do ktorých vedie cesta cez už navštívené vrcholy. Algoritmus je optimálny, teda ak medzi vrcholmi s a d cesta existuje, tak ju nájde a bude to zároveň aj najkratšia cesta. Čitateľ môže nájsť dôkaz v [Ďur]. Pseudokód tohoto algoritmu je v dodatku B.1.

V RTS však zvyčajne potrebujeme nájsť len cestu medzi dvomi vrcholmi a hľadanie všetkých ciest by bolo zbytočné. Preto môžeme Djikstrov algoritmus zastaviť, keď nájdeme cestu do cieľového vrchola.

Dijkstrov algoritmus je pre časté použitie v RTS nevhodný, keďže vždy



Obr. 3.1: Dijkstrov algoritmus

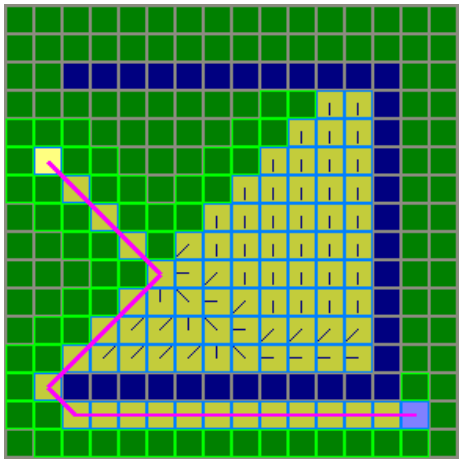
prehľadáva veľké množstvo vrcholov ako vidno na obrázku 3.1. Žlté políčka sú navštívené vrcholy (množina $\mathcal{Z}$), čierne úsečky v nich smerujú k vrcholu, cez ktorý sme do vrchola prišli a zeleným rámikom sú zvýraznené otvorené vrcholy (množina $\mathcal{O}$). Ružovou farbou je vyznačená nájdená cesta.

Môže však byť vhodný, ak chceme zistiť, ktorý z viacerých cieľov je k danému miestu najbližšie. Príkladom takejto úlohy môže byť situácia, keď chceme zistiť, ktorý z minerálov (alebo akákoľvek iná surovina v hre) je najbližší k ťažiacej jednotke.

3.2 Best-First Search

Best-First Search (BeFS) pracuje podobným spôsobom ako Dijkstrov algoritmus. Namiesto vyberania vrcholov najbližších k štartovaciemu vrchlu však BeFS vyberá vrcholy najbližšie k cieľu. To, ktorý z vrcholov je najbližšie k cieľu, vypočíta pomocou heuristickej funkcie h - t.j. rýchlo vypočítaný odhad ako ďaleko je cieľový vrchol od ľubovoľného vrchola v grafe.

Práve vďaka tomuto odhadu je oveľa rýchlejší než Dijkstrov algoritmus, keďže ten mu hovorí, ktoré vrcholy prezerať ako prvé, a preto ich nemusí prezerať tak veľa. Nevýhodou však je, že takto nájdená cesta nemusí byť najkratšia ako vidno na obrázku 3.2. Z obrázku je zrejmé, že BeFS síce neprehľadal toľko vrcholov ako Dijkstrov algoritmus v rovnakom prípade, ale v



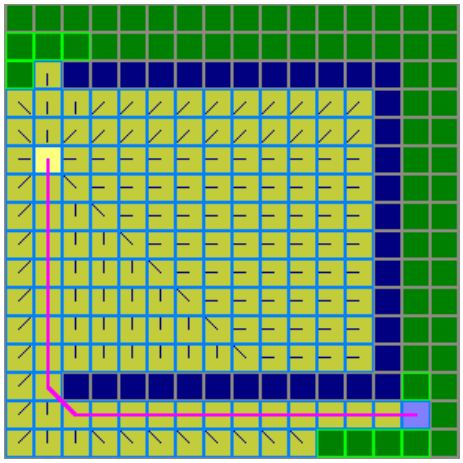
Obr. 3.2: BeFS nájde neoptimálnu cestu

prvých krokoch vybral vrcholy, ktoré na najkratšej ceste neležia a tým zišiel z optimálnej cesty.

BeFS pracuje rovnako ako Dijkstrov algoritmus. Rozdiel je len v tom, že si oproti Dijkstrovmu algoritmu pre každý vrchol v pamätá ešte aj $H[v]$ - hodnotu heuristickej funkcie pre vrchol v. BeFS si tiež udržiava množinu otvorených vrcholov $\mathcal{O}$ v prioritnom rade, ale utrieďuje ho podľa hodnôt H otvorených vrcholov a vždy vyberá ten, o ktorom si heuristika myslí, že je k cieľu najbližšie, teda vrchol v s najmenším $H[v]$. Pseudokód tohoto algoritmu je v dodatku B.2. Keďže BeFS pri výbere vrchola uvažuje len hodnotu heuristickej funkcie, môže sa stať, že vyberie vrchol, cez ktorý najkratšia cesta nevedie. Algoritmus teda nie je optimálny, ale rovnako ako Dijkstrov algoritmus, ak nejaká cesta existuje, tak ju nájde, keďže rozdiel je len v poradí, v akom vrcholy prehľadáva.

3.3 A*

Tento algoritmus vznikol v roku 1968. Je to dnes asi najpopulárnejší algoritmus na hľadanie ciest. Dôvodom je fakt, že je veľmi flexibilný a môže byť použítý v rôznych situáciach. Tak ako iné grafové algoritmy, aj A* môže prehľadať veľkú časť grafu. Môže byť použitý rovnako ako Dijkstrov algoritmus na nájdenie najkratšej cesty. Rovnako ako BeFS používa heuristickú funkciu



Obr. 3.3: AStar nájde optimálnu cestu

pri výbere vrchola, v ktorom bude prehľadávanie pokračovať.

V triviálnych prípadoch je rovnako rýchly ako BeFS. Ako vidno na obrázku 3.3, A* dokáže nájsť optimálne cesty aj v prípadoch, kde BeFS zlyhal a zároveň prehľadá menej vrcholov ako Dijkstrov algoritmus. Dôvodom je, že A* utrieďuje množinu $\mathcal{O}$ podľa hodnoty $f(v)$ vypočitanej ako

$$f(v) = g(v) + h(v),$$

kde $g(v)$ je rovnako ako pri Dijkstrovom algortime dĺžka cesty zo štartovacieho vrchola s do daného vrchola v idúc po nájdenej ceste a $h(v)$ je rovnako ako pri BeFS algoritme odhadovaná cena z vrcholu v do cieľového vrchola t.

A* tak pri výbere kombinuje informáciu, ktorú používa Dijkstrov algoritmus (preferuje vrcholy blízko štartu s) a informáciu, ktorú používa BeFS (preferuje vrcholy blízko cieľa t). Podrobný pseudokód tohoto algoritmu je v dodatku B.3. Uvádzame len kód výberu najlepšieho vrchola z množiny $\mathcal{O}$, inicializačná časť algoritmu je rovnaká ako v predchádzajúcich prípadoch. Premennú dist sme kvôli názornosti premenovali na G, keďže A* triedi množinu $\mathcal{O}$ podľa $f(v) = g(v) + h(v)$. Hlavný rozdiel oproti predchádzajúcim algoritmom je na riadku 27 v else vetve, kde A* môže presunúť uzavretý vrchol z množiny $\mathcal{Z}$ do $\mathcal{O}$.

3.3.1 Heuristika

Fungovanie algoritmu závisí od výberu heuristickej funkcie, čím môžeme kontrolovať správanie sa A*.

- Ak je $h(v) = 0 \ \forall v \in V$, potom výber ďalšieho vrchola z $\mathcal{O}$ závisí len od $g(v)$ a A* sa správa rovnako ako Dijkstrov algoritmus.
- Ak je $h(v) \leq h^*(v) \ \forall v \in V$, teda ak je heuristika h prípustná, tak je zaručené, že A* nájde najkratšiu cestu. Čím je $h(v)$ menšie, tým viac vrcholov A* preskúma, čo celý algoritmus prirodzene spomalý.
- Ak je $h(v) = h^*(v) \ \forall v \in V$, teda ak je heuristika h perfektná, tak A* pôjde presne po najkratšej ceste a nikdy z nej "neodbočí", čím bude veľmi rýchly. Hoci toto nevieme zabezpečiť vždy, v niektorých špeciálnych prípadoch sa to dá. Inými slovami, ak vieme vždy spraviť dokonalý odhad, A* sa bude správať dokonale a vždy pôjde po najkratšej ceste.
- Ak je $h(v)$ niekedy väčšia než je dĺžka z v do cieľa, tak nie je zaručené, že A* nájde najkratšiu cestu. Na druhej strane je však algoritmus rýchlejší. Príkladom je Manhattanská heuristika, pomocou ktorej je AStar rýchlejší ako Dijkstrov algoritmus.
- Ak je $h(v)$ veľmi veľké v porovnaní s $g(v)$ alebo je $g(v) = 0$, tak pri výbere ďalšieho vrchola z $\mathcal{O}$ rozhoduje $h(v)$ a A* sa správa rovnako ako BeFS algoritmus.

Ako vidno, zmenou pomeru hodnôt medzi $g(v)$ a $h(v)$ môžeme dosiahnuť rozličné výsledky a správanie sa A*. Na jednej strane môžeme rýchlo nájsť neoptimálne cesty. Naopak, ak máme dostatok času, tak môžeme získať kratšie cesty. V hrách môže byť táto vlastnosť A* veľmi užitočná a podľa danej situácie môžeme upravovať $g(v)$ alebo $h(v)$.

Na mriežkových mapách je používaných niekoľko heuristík. Uvedieme aspoň tú najznámejšiu - **Manhattanská vzdialenosť**. V princípe ide o spočítanie koľko políčok je nutné prejsť horizontálne a vertikálne z daného vrchola n do cieľa t. Nech n prislúcha políčko v hernej mape so súradnicami $[n_x, n_y]$ a t so súradnicami $[t_x, t_y]$. Potom Manhattanská vzdialenosť je určená vzťahom

$$h_m(n) = D * (|n_x - t_x| + |n_y - t_y|),$$

kde D je minimálna cena vertikálnej (resp. horizontálnej) hrany v grafe. Názov Manhattan naznačuje, že ide o počítanie počtu blokov, ktoré treba prejsť, keď sa nemôžeme pohybovať diagonálne, tak ako tomu je v Manhattane v meste New York, kde sú všetky ulice orientované rovnobežne alebo kolmo na seba. Ďalšie informácie môže čitateľ nájsť v [Pat07].

A* tak ako je popísaný v dodatku B.3 môže otvárať už uzavreté vrcholy čo môže v najhoršom prípade viesť k exponenciálnemu času. Ak však v A* použijeme namiesto prípustnej heuristiky konzistentnú, tak bude zaručené, že A* pri prvom výbere vrchola n z $\mathcal{O}$ nájde optimálnu cestu do n a nikdy nebude potrebovať n opätovne otvárať. To znamená, že pri použití konzistentnej heuristiky, nie je nutné implementovať else vetvu v algoritme na obrázku B.3.

Veta 3.3.1 *Ak je h konzistetná heuristika, tak hodnoty $f(n)$ sú pozdĺž ľubovoľnej cesty neklesajúce.*

Dôkaz. Nech je h konzistetná heuristika a nech vrchol v je následníkom vrchola u na ceste, teda $g(v) = g(u) + d(u, v)$. Potom z konzistentnosti platí

$$\begin{aligned} d(u,v) + h(v) &\geq h(u) \\ g(u) + d(u,v) + h(v) &\geq h(u) + g(u) \\ f(v) = g(v) + h(v) &\geq h(u) + g(u) = f(u) \\ f(v) &\geq f(u) \end{aligned}$$

□

Veta 3.3.2 *Ak je h konzistetná heuristika, tak je A* s takouto heuristikou optimálny t.j. $\forall n \in \mathcal{Z}\ g(n) = g^*(n)$, kde $g^*(n)$ je dĺžka najkratšej cesty zo štartového vrcholu s do n a bez nutnosti znova otvárať už uzavreté vrcholy. Teda ak A* vyberie vrchol n prvýkrát z $\mathcal{O}$, tak platí, že $g(n) = g^*(n)$.*

Dôkaz. Sporom. Nech A* práve uzavrel vrchol n taký, že $g(n) > g^*(n)$, teda $f(n)$ bolo minimálne zo všetkých vrcholov v $\mathcal{O}$ a nájdená cesta z s do n nie je optimálna. To znamená, že v grafe existuje iná cesta z s do n, ktorá je optimálna. Na tejto ceste musí byť vrchol n' z $\mathcal{O}$, pričom n je jeho následníkom. Z predchádzajucej vety však vyplýva, že $f(n') < f(n)$ čo je spor s minimalitou $f(n)$. □

Poznámka 3.3.1 Príkladom konzistentnej heuristiky je **diagonálna heuristika**

$$h_d(n) = D_d * diag(n) + D * (h_m(n) - 2 * diag(n)),$$

kde $diag(n) = min\{|n_x - t_x|, |n_y - t_y|\}$ je počet políčok, cez ktoré môžeme ísť po diagonále, D_d je cena najlacnejšej diagonálnej a D je cena najlacnejšej horizontálnej (resp. vertikálnej) hrany v grafe.

3.3.2 Modifikácie A*

Ako už bolo spomenuté, A* je veľmi flexibilný algoritmus a môže byť rôznymi spôsobmi upravený. Popíšeme si niekoľko možných variantov A*, ďalšie obmeny môže čitateľ nájsť v [Pat07].

- **Beam search**

  V $\mathcal{O}$ sú uložené všetky vrcholy, ktoré treba prehľadávať. Beam search je odlišný od klasického A* tým, že obmedzuje veľkosť tejto množiny. Keď je množina $\mathcal{O}$ príliš veľká, vrchol s najmenšou pravdepodobnosťou (najväčšia hodnota $f(v)$) na dobrú cestu je z nej odstránený.

- **Dynamic weighting**

  Tento prístup je založený na predpoklade, že na začiatku hľadania je dôležité rýchlo sa niekam pohnúť a pri konci dosiahnuť cieľ čo najlepšou cestou. V RTS vyžadujeme, aby jednotky reagovali na hráčové príkazy okamžite bez viditeľného oneskorenia. Preto môžeme výpočet $f(v)$ upraviť na

  $$f(v) = g(v) + w(v) * h(v),$$

  kde heuristickej funkcii prislúcha nejaká váhová funkcia. Myšlienka spočíva v tom, že čím bližšie sme pri cieli, tým viac znižujeme váhu. Dôsledkom toho je oslabenie významu heuristiky v prospech skutočnej dĺžky zatiaľ nájdenej cesty.

- **Bidirectional search**

  Paralelne spustíme dve prehľadávania – jedno zo štartu do cieľa a druhé z cieľa do štartu a budeme dúfať, že sa stretnú niekde v "strede" a výsledkom bude dobrá cesta. A* však neprehľadáva veľké časti mapy ako Dijkstrov algoritmus, a tak sa môže stať, že sa obe prehľadávania stretnú až v cieli a zbytočne prehľadajú veľkú časť grafu.

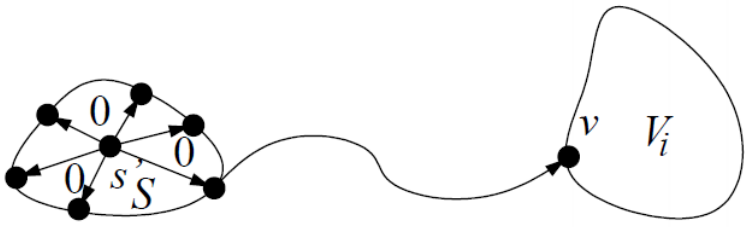
Kapitola 4

Algoritmy s predpočítanými informáciami

Algoritmy uvedené v predchádzajúcej kapitole pri hľadaní cesty nevyužívali žiadne ďalšie informácie, ktoré si môžeme o hracej mape predpočítať. Teraz si popíšeme algoritmy, ktoré hľadanie ciest urýchľujú predpočítaním si vzdialeností medzi niektorými vrcholmi, čím získajú "zmysel pre orientáciu". Ďalej navrhneme heuristiku, ktorá bude využívať informácie o danej mape tak, aby vyhľadávací algoritmus viedla pozdĺž najkratšej cesty a zohladňovala pri tom aj prekážky v mape. Pomocou tejto heuristiky sa bude dať jednoducho vypočitať horné aj dolné ohraničenie pre $d(s,t)$. Nakoniec navrhneme modifikáciu algoritmu Landmark A*.

4.1 Precomputed Cluster Distances

Precomputed Cluster Distances (PCD) [Mau06] je modifikácia Dijkstrovho algoritmu, ktorá dokáže razantne znížiť počet prehľadávaných vrcholov a tým celý algoritmus oproti klasickému Dijkstrovmu algoritmu značne urýchliť. Je založený na rozdelení grafu na k navzájom disjunktných blokov $\mathcal{V} = V_1 \cup V_2 \cup \ldots \cup V_k$, tak že každý vrchol grafu patrí do práve jedného z nich. Algoritmus pri hľadaní ciest využíva predpočítané dĺžky najkratších ciest medzi všetkými blokmi a tiež aj začiatočné a koncové vrcholy týchto ciest. Na základe týchto informácií vie vypočítať horné a dolné ohraničenia pre $d(s,t)$, ktoré sú použité na zníženie počtu prehľadávaných vrcholov.



Obr. 4.1: Výpočet vzdialenosti bloku S.

Predpočítanie informácií

Predpočítanie dĺžok najkratších ciest medzi všetkými dvojicami blokov sa dá uskutočniť k vykonaniami Dijkstrovho algoritmu. Pre blok S vypočítame jeho vzdialenosť do všetkých ostatných blokov nasledovne. Do grafu G pridáme nový vrchol s' a spojíme ho so všetkými hraničnými (vrchol susedný s vrcholom z iného bloku) vrcholmi bloku S hranami, ktorých cena je 0. Vo vrchole s' spustíme Dijkstrov algoritmus a prehľadáme celý graf G. Po skončení budeme poznať vzdialenosť S do všetkých ostatných blokov a tiež aj začiatočné a koncové vrcholy na najkratších cestách medzi nimi, pričom na uchovanie týchto informácii potrebujeme len $\mathcal{O}(k^2)$ pamäte. Obrázok[1] 4.1 znázorňuje tento postup pre blok S.

Hľadanie ciest

Pri hľadaní cesty z s do t prebieha algoritmus v dvoch smeroch v dvoch fázach.

V prvej fáze prebieha bežný obojsmerný Dijkstrov algoritmus z s do t a z t do s, až kým sa obe prehľadávania nestretnú alebo kým nie sú známe hodnoty $d(s, s')$ a $d(t, t')$, kde s' (resp. t') je hraničný vrchol najbližší k s (resp. t).

V druhej fáze popíšeme len prehľadávanie z s do t, pretože pre opačné prehľadaváníe je postup analogický. Počas prehľadávania sú prepočítavané horné ohraničenie $\hat{d}(s, t)$ pre $d(s, t)$ a dolné ohraničenie $\underline{d}(s, w, t)$ pre dĺžku ľubovoľnej cesty z s do t cez w. Prehľadávanie je optimalizované tým, že susedné vrcholy vrchola w nemusia byť vôbec prehľadané, ak $\underline{d}(s, w, t) >$

[1] Prevzaté z [JM06]

$\hat{d}(s,t)$. Inými slovami, ak najkratšia cesta z s do t cez w je dlhšia ako horné ohraničenie $d(s,t)$, tak susedov w nemusíme vôbec uvažovať, pretože w určite neleží na najkratšej ceste medzi s a t. Druhá fáza končí, keď sa prehľadávania stretnú.

Horné ohraničenie

Horné ohraničenie je prepočítané vždy, keď je nájdená najkratšia cesta do vrchola u z bloku U takého, že u je zapamätaným začiatočným vrcholom na najkratšej ceste medzi blokmi U a T. Ak t_{UT} je koncový vrchol na tejto ceste, tak pre horné ohraničenie platí vzťah

$$d(s,t) \leq d(s,u) + \underbrace{d(u,t_{UT})}_{=d(U,T)} + d(t_{UT},t).$$

Hodnota $d(s,u)$ bola práva vypočítaná a $d(u,t_{UT})$ je predpočítaná. Ak $d(t_{UT},t)$ ešte nebolo vypočítané opačným prehľadávaním, tak sa namiesto tejto hodnoty použije priemer bloku T. $\hat{d}(s,t)$ je minimálne z ohraničení určených týmto vzťahom.

Dolné ohraničenie

Dolné ohraničenie pre cestu z s do t cez vrchol w je určené vzťahom

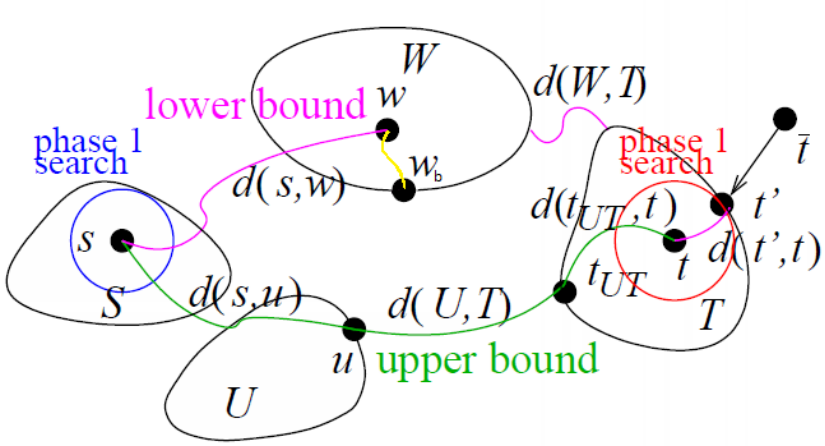
$$\underline{d}(s,w,t) = d(s,w) + d(W,T) + d(t,t').$$

Toto ohraničenie vieme vždy určiť, pretože hodnota $d(s,w)$ bola práve vypočítaná, W je pre každý vrchol w predpočítané, $d(W,T)$ je predpočítaná a $d(t,t')$ bola určená v prvej fáze. Na obrázku[2] 4.2 je znázornený výpočet oboch ohraničení.

Pamäťová zložitosť

Pri rozdelení grafu na k blokov potrebujeme $\mathcal{O}(k^2)$ pamäte na uloženie si vzájomných vzdialeností k blokov a začiatočných a koncových vrcholov na najkratších cestách medzi nimi. Pre výpočty horných a dolných ohraničení potrebuje každý vrchol vedieť, v ktorom bloku sa nachádza, na čo potrebujeme $\mathcal{O}(n)$ pamäte. Pamäťová zložitosť algoritmu je preto $\mathcal{O}(k^2+n)$. Vhodnou implementáciou môže byť znížená na $\mathcal{O}(k^2+B)$, kde B (angl. border)

[2]Prevzaté z [JM06]



Obr. 4.2: Výpočet horného a dolného ohraničenia pre $d(s,t)$.

je počet hraničných vrcholov v grafe, pretože informácia, do ktorého bloku vrchol patrí sa mení len na hraniciach blokov. Preto si stačí zapamätať id blokov všetkých B hraničných vrcholov v hašovacej tabuľke. Spracované podľa [JM06].

Zlepšenie dolného ohraničenia

Na zlepšenie kvality dolného ohraničenia využijeme nasledujúcu triviálnu úvahu. Nech bola pre vrchol w práve určená $d(s,w)$ a nech w_b je najbližší hraničný vrchol k vrcholu w. Ak sa chceme z bloku W dostať do bloku T, tak musíme blok W najskôr opustiť, a preto musíme z vrchola w prejsť aspoň vzdialenosť $d(w,w_b)$.

Dolné ohraničenie pre cestu z s do t cez vrchol w bude určené vzťahom

$$\underline{d}(s,w,t) = d(s,w) + d(w,w_b) + d(W,T) + d(t,t').$$

Aby sme takto vedeli dolné ohraničenie vypočítať, je nutné, aby každý vrchol poznal svoju vzdialenosť k jeho najbližšiemu hraničnému vrcholu, na čo potrebujeme dodatočných $\mathcal{O}(n)$ pamäte. Pamäťová zložitosť algoritmu preto zostáva $\mathcal{O}(k^2+n)$. Vzdialenosti všetkých vrcholov k ich najbližšiemu hraničnému vrcholu vieme ľahko vypočítať nasledovným spôsobom. Do grafu G pridáme vrchol s' a spojíme ho so všetkými hraničnými vrcholmi hranami,

ktorých cena je 0. Vo vrchole s' spustíme Dijkstrov algoritmus a prehľadáme celý graf G.

Na tomto mieste je tiež vhodné podotknúť, že dôsledkom tejto zmeny je odstránenie prvej fázy algoritmu. Keďže v prvej fáze algorimus hľadá k s aj t ich najbližšie hraničné vrcholy, tak ich predpočítaním sa stáva prvá fáza algoritmu úplne zbytočnou. V kapitole 5 v časti 5.4.1 sú uvedené výsledky testov porovnania algoritmu PCD a PCD so zlepšenými ohraničeniami.

4.2 Landmark A*

Heuristické funkcie spomínané v predchádzajúcej kapitole vôbec neberú do úvahy topológiu mapy a možné prekážky na nej a vychádzajú výlučne z geografických údajov (x-ová a y-ová súradnica) o cieľovom vrchole. Tieto heuristiky síce fungujú dobre na priamočiare cesty, nemajú však žiadny zmysel pre orientáciu, a tak sú ich odhady v prípade zložitých ciest príliš podhodnotené a algoritmy tak prehľadajú veľké množstvo vrcholov grafu. Ďalej, závislosť na geografických údajoch limituje ich použitie len na triedu grid grafov.

Vynikajúcim prínosom v oblasti heuristík je landmark heuristika uvedená v [AVG04]. V grafe vyberieme malú množinu významných pamiatkových (angl. landmark) vrcholov a pre každý vrchol v grafe si spočítame a zapamätáme jeho vzdialenosť do všetkých landmark-ov.

Predpočítanie informácií

Predpočítanie dĺžok najkratších ciest z každého z k landmark-ov do všetkých vrcholov sa dá jednoducho uskutočniť k vykonaniami Dijkstrovho algoritmu.

Hľadanie ciest

Landmark A* hľadá najkratšie cesty rovnakým spôsobom ako bolo popísané v 3.3. Rozdiel je len v použitej heuristickej funkcii.

Dolné ohraničenie

Dolné ohraničenia vzdialenosti dvoch vrcholov sú počítané Landmark heuristikou nasledovne. Nech $L = \{l_1, \ldots, l_k\}$ je množina landmark-ov a u a v sú ľubovoľné vrcholy v grafe. Z trojuholníkovej nerovnosti vieme, že pre dolný

odhad vzdialenosti u a v platí $abs(d(u,l) - d(v,l)) \leq d(u,v),\ \forall l \in L$. Preto pre dolné ohraničenie $d(u,v)$ môžeme použiť vzťah

$$\underline{d}(u,v) = \max_{l \in L}(abs(d(u,l) - d(v,l))).$$

Použijeme teda najtesnejšie ohraničenie spomedzi všetkých landmark-ov. Takto definovaná heuristika je konzistentná a teda aj prípustná, a preto Landmark A* vždy nájde najkratšiu cestu bez nutnosti otvárať už uzavreté vrcholy. Keďže nepoužíva ani žiadne geografické údaje o vrcholoch, je tiež použiteľná na všeobecných grafoch a nie len na grid grafoch.

Pamäťová zložitosť

Pamäťová zložitosť tohoto algoritmu je $\mathcal{O}(k.n)$, kde k je počet vybratých landmark-ov. Hoci je k konštanta, pri veľkom počte vrcholov v grafe algoritmus vyžaduje príliš veľa pamäte čo je jeho použitie v hrách obmedzuje.

Voľba landmark-ov

Výber dobrých landmark-ov je kritický pre kvalitu dolných ohraničení. Teraz si opíšeme dva spôsoby výberu landmark-ov.

Farthest Landmark (FL) - náhodne vyberieme jeden vrchol v grafe a nájdeme vrchol v_1, ktorý je od neho najďalej. Pridáme v_1 do množiny landmark-ov. Ďalej bude výber pokračovať v iteráciach, kde v každej iterácii nájdeme vrchol, ktorý je najďalej od všetkých doteraz vybratých landmark-ov a tento vrchol pridáme do množiny landmark-ov.

Optimized Farthest Landmark (OL) - vyberieme prvých k landmark-ov rovnako ako vo Farthest Landmark metóde. Potom nasleduje m iterácií, kde v každej odstránime landmark, ktorý je najbližšie k ostatným. Nový landmark vyberieme rovnakým spôsobom ako Farthest Landmark metóde.

Samozrejme existujú aj ďalšie algoritmy ako vyberať landmark-y. Založené sú na výmene najhoršieho landmark-u za najlepší z množiny kandidátov. Hoci sa v [AVG04] uvádza, že najlepšie výsledky boli dosiahnuté s landmark-ami vybratými algoritmom Optimized Planar, nebudeme ho testovať, pretože výpočet trvá na veľkých vstupoch hodiny, čo je v praxi pre bežných hráčov neprípustné. Spracované podľa [AVG04]. V kapitole 5 v časti 5.4.3 sú uvedené výsledky testov porovnania algoritmu Landmark A* s ostatnými algoritmami.

4.3 Waypoint A*

Podobne ako PCD, aj Waypoint A* využíva rozdelenie grafu na k navzájom disjunktných blokov $\mathcal{V} = V_1 \cup V_2 \cup \ldots \cup V_k$. V celom grafe sa vyberie k waypoint vrcholov a vypočíta sa ich vzájomná vzdialenosť. Pre každý vrchol sa nájde k nemu najbližší waypoint vrchol a tým sa celý graf rozdelí na k disjunktných blokov. Na základe týchto informácií sa pre ľubovoľnú dvojicu vrcholov dá vypočítať dolné ohraničenie ich vzdialenosti, ktoré sa použije ako hodnota heuristickej funkcie pri hľadaní najkratšej cesty.

Predpočítanie informácií

Vypočítať vzájomné vzdialenosti všetkých waypoint-ov vieme spraviť vykonaním k prehľadávaní grafu Dijkstrovým algoritmom postupne z každého waypoint-u. Vzdialenosť všetkých vrcholov od ich najbližšieho waypoint vrcholu zistíme nasledovne. Do grafu pridáme vrchol s' a spojíme ho so všetkými waypoint vrcholmi hranami s cenou 0. Vo vrchole s' spustíme Dijkstrov algoritmus a po jeho skončení budeme poznať požadované vzdialenosti.

Hľadanie ciest

Waypoint A* hľadá najkratšie cesty rovnakým spôsobom ako bolo popísané v 3.3. Rozdiel je len v použitej heuristickej funkcii.

Dolné ohraničenie

Definícia 4.3.1 Nech u a v sú ľubovoľné vrcholy v grafe a nech w_u a w_v sú ich najbližšie waypoint vrcholy. **Waypoint heuristika** $h_w(n)$ bude definovaná ako

$$h_w(n) = \begin{cases} d(w_u, w_v) - d(u, w_u) - d(v, w_v), & d(u, w_u) + d(v, w_v) < d(w_u, w_v) \\ h_d(n), & inak \end{cases},$$

kde $d(w_u, w_v)$ je vzdialenosť waypoint vrcholov w_u a w_v, $d(u, w_u)$ je vzdialenosť vrchola u do jeho najbližšieho waypoint vrchola w_u a $d(v, w_v)$ je vzdialenosť vrchola v do jeho najbližšieho waypoint vrchola w_v. Ak je $d(u, w_u) + d(v, w_v) \geq d(w_u, w_v)$, tak použijeme diagonálnu heuristiku $h_d(n)$, ktorá je prípustná.

Takto definovaná heuristika je prípustná, ale nie je konzistentná. Ak by sme vyžadovali, aby A* s touto heuristkou vždy našiel optimálnu cestu, museli by sme použiť A*, ktorý môže otvárať už uzavreté vrcholy. Pseudokód je rovnaký ako v prípade už uvedeného A* algoritmu, rozdiel je len v použitej heuristickej funkcii, a preto ho neuvádzame.

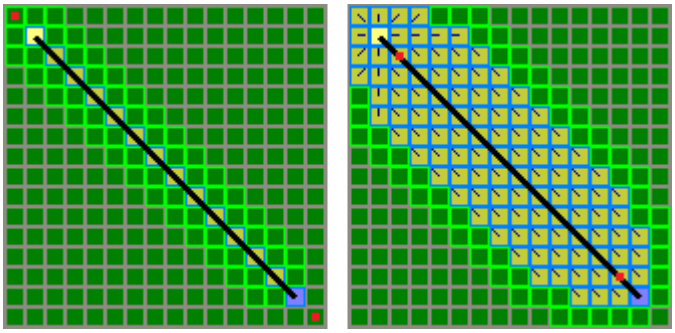
Pamäťová zložitosť

Na uchovanie vzájomných vzdialeností k waypoint-ov potrebujeme $\mathcal{O}(k^2)$ pamäte. Ďalej, každý vrchol si musí pamätať, do ktorého bloku patrí a tiež vzdialenosť do najbližšieho waypoint vrcholu. Preto je pamäťová zložitosť algoritmu $\mathcal{O}(k^2 + n)$.

4.4 Landmark A* Bounded

Ako bolo konštatované v [AVG04], pre grafy v ktorých sú vzdialenosti vrcholov silno korelované so vzdialenosťami vrcholov v nákresoch grafov v rovine, Landmark heuristika vypočíta dobré odhady, ak sa landmark nachádza za cieľovým vrcholom.

Pre ilustráciu uvádzame obrázok 4.3, kde sú landmark-y označené červenou, štartovacie políčko žltou a cieľové modrou farbou. Na prvý pohľad je zrejmé, že v takmer identických situáciach algoritmus prehľadá viac vrcholov, ak landmark nie je umiestnený "za" cieľovým vrcholom.



Obr. 4.3: Prehľadané vrcholy s rozdielným umiestnením landmark-ov

Keďže pamäťová zložitosť algoritmu Landmark A* je $\mathcal{O}(k.n)$, tak nemôžeme zvoliť k príliš veľké. Potom sa však landmarky geometricky nachádzajú

len za malým počtom vrcholov. Ako vidno z výsledkov testov uvedených v kapitole 5 v časti 5.4.3, s nízkym počtom landmark-ov Landmark A* v typoch testov BFS a FFL prehľadáva veľkú časť grafu, pretože odhady vzdialeností vypočítané heuristickou funkciou sú príliš nízke.

Preto sa pokúsime navrhnúť a experimentálne porovnať modifikáciu Landmark A* algoritmu Landmark A* Bounded, ktorá bude používať $\mathcal{O}(n)$ pamäte a zároveň bude porovnateľne rýchla ako pôvodný Landmark A* so 16 landmark-ami.

Skombinujeme výhody všetkých predchádzajúcich algoritmov - z Landmark A* konzistentnú heuristiku, ktorá je tým presnejšia čím sú s a t bližšie k landmark-om a z PCD a Waypoint A* ich horné a dolné ohraničenia, ktoré budú Landmark A* v prípade potreby obmedzovať a nedovolia pri nízkych odhadoch landmark heuristiky prehľadávať vrcholy, o ktorých PCD a Waypoint A* vedia, že cez ne najkratšia cesta nevedie. Keďže Landmark heuristika je konzistentná, tak tento modifikovaný algoritmus nájde najkratšie cesty bez nutnosti otvárať už uzavreté vrcholy.

Predpočítanie informácií

Predpočítanie informácií sa vykoná rovnakým postupom ako to bolo popísané v algoritmoch PCD, Landmark A* a Waypoint A*.

Hľadanie ciest

Algoritmus pri hľadaní najkratších ciest používa rovnakú heuristickú funkciu ako Landmark A*. Naviac však využíva myšlienku uvedenú v PCD – ak je dolné ohraničenie $\underline{d}(s, w, t)$ cesty z s do t cez vrchol w väčšie ako horné ohraničenie $d(s, t)$, tak susedov w neprehľadáva.

Horné ohraničenie

PCD horné ohraničenie je vypočítané rovnakým spôsobom ako bolo napísané v časti 4.1. Narozdiel od PCD, Waypoint heuristikou vieme vypočítať horné ohraničenie pre každý vrchol u nasledovne. Nech w_u je najbližší waypoint vrchol k u a w_t k t. Potom

$$d(s, t) \leq d(s, u) + d(u, w_u) + d(w_u, w_t) + d(w_t, t).$$

Toto ohraničenie vieme vždy určiť, pretože hodnota $d(s, u)$ bola práva vypočítaná a ostané hodnoty sú predpočítané. Horné ohraničenie $\hat{d}(s, t)$ je menšie z dvojice PCD a waypoint ohraničení.

Dolné ohraničenie

Dolné ohraničenia sú vypočítané rovnakým spôsobom ako bolo napísané v časti 4.1 a 4.3. Jedinou zmenou je, že k hodnote waypoint heuristiky $h_w(n)$ pripočítame $d(s, n)$, keďže heuristika dáva odhad vzdialenosti z n do t a my požadujeme dolné ohraničenie $d(s, t)$.

Pamäťová zložitosť

Nech je v grafe zvolených k waypoint-ov a l landmark-ov. Keďže algoritmus je len kombináciou predchádzajúcich, tak jeho pamäťová zložitosť je $\mathcal{O}(k^2+l.n)$. Ak bude $k \leq \sqrt{n}$, tak pamäťová zložitosť bude lineárna.

Kapitola 5

Experimenty

Všetky algoritmy boli implementované v programovacom jazyku C++ použitím dátových štruktúr vector, queue a priority_queue z STL a QThread z knižnice QT v4.6.2 a skompilované v Microsoft Visual C++ 2008 Express Edition. Všetky testy boli vykonané na počítači so štvorjadrovým procesorom Intel Q9450 2,66GHz so 4GB RAM v operačnom systéme Microsoft Windows7 64bit v konzolovej release verzii programu PathfinderCmd.exe. Každý test bol vykonaný 5 krát a pri porovnávaní algoritmov bol použitý priemer týchto 5 testov. Všetky časy boli merané QueryPerformanceCounter funkciou. Pri bidirectional (BD) algoritmoch budeme používať ich multithreaded (MT) verzie, kde vyhľadávanie v každom smere prebieha v samostatnom vlákne.

5.1 Generovanie náhodných máp

Hracie mapy v RTS nie sú náhodné, ale snažia sa pripomínať skutočný realistický terén. V praxi sú takéto mapy často reprezentované pomocou heightmáp. Heightmapy sú dvojrozmerné polia čísel, kde číslo na pozícii [x, y] predstavuje výšku terénu v bode [x, y]. Zvyčajne sú všetky čísla z intervalu $\langle -1.0, 1.0 \rangle$, pričom -1.0 znamená najmenšiu a 1.0 najväčšiu výšku. Dôležité pre výsledny vzhľad mapy je, aby boli všetky hodnoty v heightmape náhodné, ale zároveň, aby sa čísla na susedných pozíciach príliš nelíšili.

Všetky testované mapy boli generované pomocou open-source knižnice libnoise. Táto knižnica generuje tzv. "súvislý šum" (angl. coherent noise), čo je hladko sa meniaci šum. Všetky mapy majú rozmer 1024x1024.

Typy terénu

V testovaných mapách rozlišujeme štyri druhy terénu - voda, tráva, hlina, skaly. Prechod je dovolený len medzi vrcholmi s terénom typu tráva, hlina a skaly. Mapy sú tiež generované tak, aby všetky tieto vrcholy boli v jednom komponente.

Typy hrán

V testovaných mapách rozlišujeme diagonálne a horizontálne (resp. vertikálne) hrany. Prechod po diagonálnej hrane je dlhší ako po horizontálnej a vertikálnej, čo vyjadríme vyšším ohodnotením hrany. Ak je dĺžka horizontálnej (resp. vertikálnej) hrany 1, tak dĺžka diagonálnej je približne $\sqrt{2} = 1,4$. Keďže počítače pracujú s celými číslami rýchlejšie a presnejšie ako s desatinnými, priradíme horizontálnym a vertikálnym hranám dĺžku 10 a diagonálnym 14, čo dostatočne odráža ich skutočný pomer.

Cena hrán

Cena hrany závisí od jej dĺžky a typu terénu oboch vrcholov, ktoré spája. Pre každý typ terénu definujeme v tabuľke 5.1 konštantu, ktorá vyjadruje náročnosť pohybu po tomto teréne. Pri výpočte ceny hrany použijeme súčet náročností oboch vrcholov vynásobený dĺžkou hrany.

Postupnosť krokov generovania máp

Všetky mapy boli vygenerované programom MapGenerator.exe, ktorý sa nachádza na priloženom DVD v adresári Tools. Pri všetkých testovaných mapách uvádzame všetky potrebné údaje, aby sa dali znovu vygenerovať. Generovanie heightmapy prebieha v týchto krokoch:

1. Nastavenie potrebných parametrov knižnice libnoise.
2. Vygenerovanie heightmapy.
3. Namapovanie čísel heightmapy na typy terénov podľa tabuľky 5.1.
4. Nájdenie všetkých komponentov podľa typu terénu.
5. Zaplnenie všetkých komponentov okrem najväčšieho vodou.

Tabuľka 5.1: Náročnosti pohybu a rozpätie hodnôt pre typy terénov

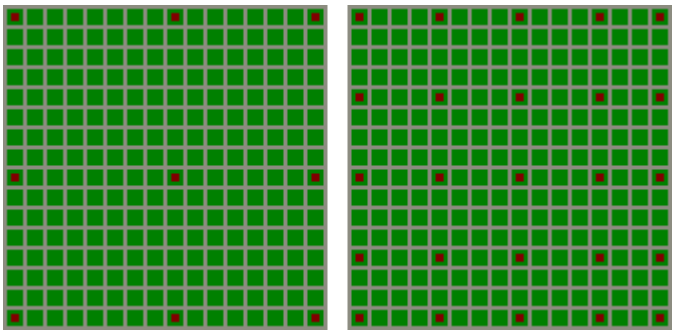
Typ terénu	Náročnosť pohybu	Rozpätie hodnôt v heightmape
voda	10	$\langle -1.0, -0.5)$
tráva	10	$\langle -0.5, 0.25)$
hlina	20	$\langle 0.25, 0.75)$
skaly	40	$\langle 0.75, 1.0\rangle$

5.2 Výber waypoint-ov

V tejto časti popíšeme spôsob výberu k waypoint-ov v grafe. Waypoint-y budeme umiestňovať tak, aby na mape tvorili mriežku, pričom so zväčšujúcim sa k bude mriežka hustejšia.

Do každého rohu vždy umiestnime jeden waypoint a zostávajúce waypoint-y rozmiestnime rovnomerne medzi nimi. Počet waypoint-ov v riadku (resp. stĺpci) budeme označovať ako hustota waypoint-ov. Na obrázku 5.1 je znázornené rozloženie waypoint-ov na mape s 256 vrcholmi s hustotou 3 a 5. Podrobný pseudokód výberu waypoint-ov čitateľ nájde v dodatku B.4.

Keďže pamäťová zložitosť algoritmov PCD, Waypoint A* je $\mathcal{O}(k^2 + n)$, tak zvolením $k = 1024$ waypoint-ov na mapách s rozmermi 1024x1024 bude pamäťová zložitosť lineárna. Testovať budeme rozloženie waypoint-ov s hustotou 16 a 32. Očakávame, že s väčšou hustotou waypoint-ov sa budú namerané časy algoritmov zlepšovať. PCD bude využívať rovnaké rozdelenie na bloky ako Waypoint A*.



Obr. 5.1: Rozloženie waypoint-ov s hustotou 3 a 5 na mape s 256 vrcholmi

5.3 Generovanie testov

Vrcholy s a t sú vždy vyberané z množiny vrcholov s terénom typu tráva, hlina a skaly. Tieto vrcholy sa vždy nachádzajú v rovnakom komponente grafu, a preto medzi nimi vždy existuje cesta. V tejto práci sa venujeme štyrom typom testov. Na každej z 10 máp vygenerujeme rovnaký počet testov.

Random (RND). Náhodne vyberieme 100 dvojíc vrcholov s a t so všetkých vrcholov. Cieľom testov je porovnať algoritmy na náhodných dvojiciach vrcholov bez akýchkoľvek kritérií.

Breadth-First Search (BFS). Náhodne vyberieme s so všetkých vrcholov, pustíme BFS prehľadávanie v s a nájdeme všetky vrcholy, ktoré sú od s vzdialené h hrán a z nich vyberieme t náhodne. Postupne vygenerujeme 50 dvojíc pre $h \in \{250, 700, 950\}$. Cieľom testov je porovnať algoritmy pri hľadaní rôzne dlhých ciest a pozorovať ako sa na ich výkone prejaví rôzny počet waypoint-ov a landmark-ov.

Far From Landmarks (FFL). Pre danú množinu 16 landmark-ov vybratých OL algoritmom vyberieme s náhodne tak, aby bol aspoň 200 hrán od všetkých landmark-ov. V s spustíme BFS prehľadávanie a nájdeme všetky vrcholy, ktoré sú od s vzdialené 950 hrán a z nich vyberieme t náhodne tak, aby bol aspoň 200 hrán od všetkých landmark-ov. Ďalej budeme znižovať počet landmark-ov na 8, 4, 2 a 1 a budeme sledovať ako sa toto prejaví na výkone algoritmov. Cieľom testov je ukázať, že so znižovaním počtu landmark-ov prudko klesá výkon Landmark A*.

Close To Landmarks (CTL). Z danej množiny 16 landmark-ov vybratých OL algoritmom vyberieme náhodne dva rôzne landmark-y l_1 a l_2. Potom vyberieme s (resp. t) náhodne tak, aby bol najviac 50 hrán od l_1 (resp. od l_2) a zároveň aby l_1 (resp. l_2) ležal z pohľadu t (resp. s) geometricky za s (resp. t). Takto volené s a t tvoria veľmi ľahké problémy pre Landmark A*. Cieľom testov je ukázať, že Landmark A* (a tiež aj Bounded) v ideálnych prípadoch prehľadá len veľmi malú časť grafu a je lepší ako PCD.

5.4 Výsledky experimentov

Výsledky všetkých testov sa nachádzajú v adresári Results na priloženom DVD. Pre každú mapu sú výsledky uvedené v samostatnom súbore. Záložka Sheet obsahuje dáta zo vstupného xml súboru. V stĺpci A je meno mapy, v B typ testu, v C meno testu, v D meno súboru s testovanými dvojicami s a t, v E meno súboru s waypoint-ami a predpočítanými dátami, v F meno súboru s landmark-ami a predpočítanými dátami. V stĺpci G sú postupne pre dvojice (waypoint súbor, landmark súbor) vypísané v riadkoch všetky testované dvojice s a t pre všetky testované algoritmy. V stĺpcoch I, Q, Y, AG, AO, AW, BE sa v každom riadku nachádzajú namerané hodnoty pre danú testovanú dvojicu a algoritmus.

Hodnoty sú v poradí čas testu v milisekundách, počet prehľadaných vrcholov, počet krokov algoritmu, počet vrcholov v množine $\mathcal{O}$ po skončení algoritmu, počet vrcholov v množine $\mathcal{Z}$ po skončení algoritmu, čas testu v mikrosekundách a dĺžka nájdenej cesty v krokoch. Sú tu uvedené hodnoty z 5 pokusov a tiež priemer týchto hodnôt. Pre každú dvojicu (waypoint súbor, landmark súbor) sú po všetkých testovacích dvojiciach pre každý algoritmus uvedené priemerné hodnoty zo všetkých priemerov testovaných dvojíc. Tieto priemerné hodnoty sú zoskupené v tabuľkách a graf na ďalších záložkách.

V súbore Average.xlsx sú uvedené priemerné hodnoty z priemerov zo všetkých 10 testovaných máp. V tejto práci sú uvedené práve tieto hodnoty. Porovnané boli algoritmy PCD, PCD so zlepšenými dolnými ohraničeniami, Landmark A* a Landmark A* Bounded. Všetky testy boli vykonané s hustotou 8 a 16 waypoint-ov pre 1, 2, 4, 8 a 16 landmark-ov generovaných OL algoritmom.

5.4.1 PCD vs PCD Imp

Algoritmus s novými dolnými ohraničeniami prehľadá v priemere o 10%-13% vrcholov menej a čas behu je o 6%-11% kratší ako pôvodný PCD algoritmus. So zvyšujúcim sa počtom blokov zrýchlenie klesá, pretože pre nové dolné ohraničenia platí nasledovné: čím je vrchol ďalej od najbližšieho hraničného vrchola, tým sú dolné ohraničenia kvalitnejšie. S väčším počtom blokov v grafe majú ale bloky menší priemer, vrcholy už nie sú tak vzdialené od hraníc blokov, a preto kvalita dolných ohraničení mierne klesá čo sa prejaví v počte prehľadaných vrcholov a následne aj v časoch.

V dodatku C sú tabuľky s údajmi z meraní v jednotlivých testoch. Ta-

buľky C.1 - C.4 obsahujú údaje podľa počtu prehľadaných vrcholov a tabuľky C.5 - C.8 podľa nameraných časov. V riadkoch PCD a PCD Imp sú uvedené prehľadané vrcholy (resp. namerané časy v ms), v treťom riadku je zrýchlenie a v poslednom je počet % o koľko menej bolo prehľadaných vrcholov (resp. bol čas behu kratší). Zvyšujúci sa počet landmark-ov na tieto algoritmy nemá vplyv, a preto podľa očakávania je množstvo prehľadaných vrcholov (resp. časy) rovnaké.

5.4.2 Waypoint A* vs PCD

Ako už bolo spomenuté, waypoint heurisitka nie je konzistentná, a preto musí byť A* umožnené otvárať uzavreté vrcholy, aby bolo zaručené, že nájde najkratšiu cestu. Podľa očakávania má Waypoint A* výrazne horšie časy, keďže musel vykonať väčší počet krokov. Časy PCD sú v priemere o 38% - 88% kratšie ako časy Waypoint A*. V dodatku C sú v tabuľke C.9 uvedené časy pre typ testu BFS 950.

Zaujímavým výsledkom ale je počet prehľadaných vrcholov, ktorý je menší ako pri jednosmernom PCD čo naznačuje, že ohraničenia vypočítané waypoint heuristikou sú porovnateľné s PCD ohraničeniami a môžu byť použité v Landmark A* Bounded na redukovanie počtu prehľadávaných vrcholov. Waypoint A* v priemere prehľadal o 43% - 63% vrcholov menej ako PCD. V dodatku C sú v tabuľke C.10 uvedené počty prehľadaných vrcholov pre typ testu BFS 950.

Keďže časy pre ostatné typy testov sú veľmi podobné, tak ich neuvádzame. Na priloženom DVD sa v súbore Results\PCDvsWaypoint\Average.xlsx nachádzajú výsledky pre všetky typy testov spolu s grafmi. Podľa očakávania sa v oboch algoritmoch so zvyšujúcim sa počtom blokov znižuje čas, počet krokov a aj počet prehľadaných vrcholov.

5.4.3 Landmark A* vs PCD

V tejto časti zanalyzujeme výsledky pre typy testov RND, BFS, FFL a CTL. Ku všetkým typom testov uvádzame tiež aj grafy s časmi všetkých algoritmov. V grafoch sú na y-ovej súradnici namerané časy v ms, W<x> OL<y> na x-ovej súradnici znamená, že hustota waypoint-ov bola x a počet landmark-ov y. Všetky prezentované údaje sa nachádzajú na priloženom DVD. Výsledky pre typy testov RND, BFS a FFL sú v súbore Results\RND-BFS-FFL\Average.xlsx a pre CTL v súbore Results\CTL\Average.xlsx.

RND

Ako najlepší algoritmus z testov vychádza Landmark A* Bounded. Podľa očakávania sa PCD algoritmy so zvyšujúcim počtom blokov zlepšujú. Grafy nameraných časov sú na obrázku C.1.

Časy Landmark A* Bounded sú s hustotou waypoint-ov 16 v priemere kratšie ako PCD o 38% - 59% a s hustotou 32 o 31% - 49%. V porovnaní s Landmark A* sú časy kratšie s hustotou 16 o 4% - 15% a s hustotou 32 o 12% - 23%, čo potvrdzuje predpoklad, že PCD a waypoint ohraničenia redukujú počet prehľadávaných vrcholov.

V prípade keď bol použitý len jeden landmark bol čas stále lepší ako PCD, čo tiež potvrdzuje predpoklad, že tieto ohraničenia nedovolia Landmark A* Bounded prehľadávať veľkú časť grafu, ak sú jeho odhady príliš nízke.

BFS

Z výsledkov získaných z testov typu BFS vyplýva, že s narastajúcou vzdialenosťou medzi s a t potrebuje Landmark A* stále viac landmark-ov na to, aby prekonal PCD. Pre vzdialenosti 250 hrán je lepší so 4, pre 700 s 8 a pre 950 sa so 16 landmark-ami k PCD len tesne približuje. Dôvodom môže byť fakt, že s narastajúcou vzdialenosťou sa s aj t nachádzajú bližšie pri okraji mapy a landmark-y generované metódou optimized farthest sa nachádzajú v maximálnej možnej vzdialenosti, teda tiež blízko krajov mapy. Keďže v grafe je 1024^2 vrcholov a landmark-ov je len 16, tak pravdepodobnosť, že s a t budú vhodne umiestnené, je nízka. Grafy na obrázkoch C.2, C.3 a C.4 zobrazujú časy pre typy testov BFS s $h \in \{250, 700, 950\}$. Pre vzdialenosti 250 hrán je z testov najlepší Landmark A* Bounded, pri ostatných sú časy veľmi blízke PCD. Landmark A* Bounded je tiež najlepší vo všetkých prípadoch podľa počtu prehľadaných vrcholov.

FFL a CTL

Testy FFL a CTL skončili presne podľa očakávania. Grafy nameraných časov sú na obrázku C.5. FFL testy boli generované tak, aby boli ťažkými problémami pre Landmark A* čo sa aj potvrdilo. Všetky ostatné algoritmy sú v týchto testoch lepšie. So 16 landmark-ami je Landmark A* 2 násobne pomalší a s klesajúcim počtom landmark-ov sa jeho časy ďalej zhoršujú až 9 násobne. Presné údaje sú uvedené v tabuľke C.11 v dodatku C. Keďže Land-

mark A* Bounded má rovnaké časy ako PCD, tak môžeme usudzovať, že tieto problémy rieši PCD lepšie ako Landmark.

V testoch CTL je situácia opačná. Tieto testy boli generované tak, aby boli ideálnymi problémami pre Landmark A*. Z testov vyplýva, že časy Landmark A* sú v priemere o 67% - 93% kratšie ako PCD a zrýchlenie je 3 - 15 násobné. Podrobnejšie údaje čitateľ nájde v dodatku C v tabuľke C.12.

5.5 Odporúčania

Na základe výsledkov testov môžeme navrhnúť niekoľko odporúčaní ako efektívne hľadať cesty. Ako sa ukázalo, Landmark A* má svoje výhody aj nevýhody. V preňho ideálnych prípadoch je veľmi rýchli. Existuje však veľa problémov, na ktorých je s malým množstvom landmark-ov výrazne pomalší ako ostatné algoritmy.

Vyhľadávanie ciest je komplexný problém, ktorý sa zrejme nedá riešiť len jedným algoritmom. Pred každým vyhľadávaním je preto dobré spraviť rýchlu analýzu a použiť algoritmus, ktorý danú situáciu rieši najlepšie. Ak sú s a t blízko landmark-ov, je najlepšou voľbou Landmark A*. Naopak, ak sú príliš vzdialené, tak je dobrou voľbou Landmark A* Bounded. Keďže sú dnes viacjadrové procesory bežné, oplatí sa tiež používať MT verzie algoritmov a skutočne tak využiť potenciál bidirectional algoritmov. Treba však podotknúť, že multithreading je výhodne použiť len na hľadanie dlhých ciest, pretože réžia potrebná na naštartovanie a synchronizáciu vlákien na krátkych cestách preváži prínos multithreadingu.

Kapitola 6

Záver

V práci sme sa venovali vyhľadávaniu najkratších ciest v 2D mapách, pričom sme algoritmom dovolili používať predpočítané informácie o mape. Jediné obmedzenie bolo, aby pamäťová zložitosť algoritmov bola $\mathcal{O}(n)$.

Najskôr sme si vysvetlili základy RTS a definovali niekoľko základných pojmov. Stručne sme sa oboznámili s najznámejšími a najpoužívanejšími grafovými algoritmami, pričom sme sa podrobnejšie venovali algoritmu A* a heuristikám.

V kapitole 4 sme sa venovali dvom novým algoritmom z posledných rokov – PCD a Landmark A*. Navrhli sme zlepšenie dolných ohraničení v PCD použitím $\mathcal{O}(n)$ pamäte a zaviedli sme tiež novú waypoint heuristiku, ktorá je síce iba prípustná, ale zohľadňuje topológiu grafu. Nakoniec sme navrhli modifikáciu algoritmu Landmark A* tak, aby využívala výhody PCD a aj s menším počtom landmark-ov ponúkala uspokojivé výsledky.

Algoritmy sme implementovali a prakticky otestovali a porovnali. V kapitole 5 sme najskôr popísali spôsob generovania máp, výber waypoint-ov a rozdelenie grafu ma disjunktné bloky. Potom sme popísali generovanie jednotlivých typov testov a ich očakávané výsledky. Nakoniec sme sa v časti 5.4 venovali analýze výsledkov jednotlivých testov a na ich základe sme sformulovali niektoré odporúčania pre efektívne hľadanie ciest.

Dodatok A

Obsah priloženého DVD

Na priloženom DVD je uložená táto diplomová práca v elektronickej podobe, aplikácia vizualizujúca implementované algoritmy, zdrojový kód programu, testovacie dáta a tiež výsledky všetkých testov.

Tabuľka A.1: Adresár /Bin.

Názov súboru	Popis
Gatial.pdf	Táto práca vo formáte PDF.

Tabuľka A.2: Adresár /Other.

Názov súboru	Popis
tinyxml.zip	XML knižnica použitá v konzolovej verzii programu na načítanie vstupného XML súboru.
qt-win-opensource-4.6.2-vs2008.exe	GUI knižnica.

Tabuľka A.3: Adresár /PathFinder – spustiteľný program.

Názov súboru	Popis
PathFinder.exe	GUI verzia aplikácie.
PathFinderCmd.exe	Konzolová verzia aplikácie.
PathFinderPrecompute.exe	Aplikácia na predpočítanie informácií.

Tabuľka A.4: Adresár /Results – výsledky testov.

Názov adresára	Popis
CTL	Výsledky testov typu CTL.
PCDvsWaypoint	Výsledky testov PCD a Waypoint A*.
RND-BFS-FFL	Výsledky testov typu RND, BFS a FFL.

Tabuľka A.5: Adresár /Src – zdrojový kód programu.

Názov súboru	Popis
src.zip	Obsahuje zdrojový kód programu. Skomprimované programom WinZip.
src.rar	Obsahuje zdrojový kód programu. Skomprimované programom WinRar.

Tabuľka A.6: Adresár /TestMaps – mapy, waypoint-y, landmark-y, testy.

Názov súboru	Popis
/1024/1024.rar	Mapy a všetky predpočítané dáta.

Tabuľka A.7: Adresár /Tools – pomocné programy.

Názov súboru	Popis
MapGenerator.exe	Program na generovanie testovacích máp.
MapGenerator.rar	Zdrojový kód programu.

Dodatok B

Pseudokódy algoritmov

```
1  PriorityQueue open = {startNode};
2  for each vertex v in Graph {
3    dist[v] := infinity;
4    previous[v] := NULL;
5  }
6  dist[startNode] := 0;
7
8  while (!open.empty()) {
9    // Vyberieme vrchol u s najmenšou dist[u]
10   u := open.remove_front();
11
12   // Vrchol môže byť v open niekoľko krát
13   // Ak už bol uzavretý, ignorujeme ho
14   if (!u.closed) {
15     u.closed := true;
16
17     // Algoritmus končí, keď sme vybrali cieľový vrchol
18     if (u == destNode) {
19       return;
20     }
21
22     // Prejdeme cez všetkých susedov vrchola u
23     for each neighbor v of u {
24       // Prejdeme len cez neuzatvorených susedov
25       if (!v.closed) {
26         // Dĺžka cesty do vrchola v cez vrchol u
27         alt := dist[u] + edge(u, v).cost;
28         // Ak je táto cesta kratšia ako súčasná
29         if (alt < dist[v]) {
30           dist[v] := alt;
31           previous[v] := u;
32           open.push(v);
33         }
34       }
35     }
36   }
37 }
```

Obr. B.1: Pseudokód Dijkstrovho algoritmu

```
1  PriorityQueue open = {startVertex};
2  for each vertex v in Graph {
3    dist[v] := infinity;
4    H[v] := infinity;
5    previous[v] := NULL;
6  }
7  dist[startNode] := 0;
8  H[startNode] := 0;
9
10 while (!open.empty()) {
11   // Vyberieme vrchol u s najmenšou H[u]
12   u = open.remove_front();
13
14   // Vrchol môže byť v open niekoľko krát
15   // Ak už bol uzavretý, ignorujeme ho
16   if (!u.closed) {
17     u.closed := true;
18
19     // Algoritmus končí, keď sme vybrali cieľový vrchol
20     if (u == destNode) {
21       return;
22     }
23
24     // Prejdeme cez všetkých susedov vrchola u
25     for each neighbor v of u {
26       // Prejdeme len cez neuzatvorených susedov
27       if (!v.closed) {
28         alt := dist[u] + edge(u, v).cost;
29         // Ak je táto cesta kratšia ako súčasná
30         if (alt < dist[v]) {
31           // Vzdialenosť vrchola v a cieľového vrchola
32           // vypočítaná heuristickou funkciou h
33           H[v] := h(v);
34           dist[v] := alt;
35           previous[v] := u;
36           open.push(v);
37         }
38       }
39     }
40   }
41 }
```

Obr. B.2: Pseudokód BFS algoritmu

```
1  while (!open.empty()) {
2    // Vyberieme vrchol u s najmenšou F[u] = G[u] + H[u]
3    u = open.remove_front();
4
5    // Vrchol môže byť v open niekoľko krát
6    // Ak už bol uzavretý, ignorujeme ho
7    if (!u.closed) {
8      u.closed := true;
9
10     // Algoritmus končí, keď sme vybrali cieľový vrchol
11     if (u == destNode) {
12       return;
13     }
14
15     // Prejdeme cez všetkých susedov vrchola u
16     for each neighbor v of u {
17       // Ak nie je uzavretý
18       if (!v.closed) {
19         alt := G[u] + edge(u, v).cost;
20         // Ak je táto cesta kratšia ako súčasná
21         if (alt < G[v]) {
22           H[v] := h(v);
23           G[v] := alt;
24           previous[v] := u;
25           open.push(v);
26         }
27       } else {
28         // Otvoríme ho, ak je cesta do v cez u lepšia
29         alt := G[u] + edge(u, v).cost;
30         if (alt < G[v]) {
31           v.closed := false;
32           H[v] := h(v);
33           G[v] := alt;
34           previous[v] := u;
35           open.push(v);
36         }
37       }
38     }
39   }
40 }
```

Obr. B.3: Pseudokód A* algoritmu bez inicializačnej časti

```
int x = 0;
int y = 0;
int space = (width - density) / (density - 1);
int remaining_space = width - density - space * (density - 1);
int pad_x = remaining_space;
int pad_y = remaining_space;

for (int i = 0; i < density; i++)
{
    if (i == density - 1)
    {
        y = height - 1;
    }

    AddWaypoint(GetNode(0, y));

    x = 0;
    pad_x = remaining_space;
    for (int j = 0; j < density - 2; j++)
    {
        x += space + 1;
        if (pad_x > 0)
        {
            x += 1;
            pad_x--;
        }

        AddWaypoint(GetNode(x, y));
    }

    AddWaypoint(GetNode(width - 1, y));

    y += space + 1;
    if (pad_y > 0)
    {
        y += 1;
        pad_y--;
    }
}
```

Obr. B.4: Pseudokód generovania waypoint-ov

Dodatok C

Tabuľky a grafy nameraných hodnôt

Tabuľka C.1: PCD vs PCD Imp - prehľadané vrcholy pre test RND

RND100	W16 OL1	W16 OL2	W16 OL4	W16 OL8	W16 OL16
PCD	34437,1134	34433,2902	34432,5824	34437,3266	34435,7526
PCD Imp	29782,546	29782,467	29783,8874	29784,5858	29783,3916
Speedup	1,156285074	1,15615977	1,156080868	1,156213044	1,156206555
(%)	13,52%	13,51%	13,50%	13,51%	13,51%
RND100	W32 OL1	W32 OL2	W32 OL4	W32 OL8	W32 OL16
PCD	23605,106	23604,3928	23604,1058	23602,9376	23606,588
PCD Imp	21049,9012	21049,7156	21049,4206	21047,0532	21051,023
Speedup	1,121387971	1,121363977	1,121366058	1,121436686	1,121398613
(%)	10,82%	10,82%	10,82%	10,83%	10,83%

Tabuľka C.2: PCD vs PCD Imp - prehľadané vrcholy pre test BFS 250

BFS250	W16 OL1	W16 OL2	W16 OL4	W16 OL8	W16 OL16
PCD	30488,4356	30490,9496	30485,6972	30487,9324	30489,7604
PCD Imp	27220,1808	27223,1508	27224,0632	27228,8712	27218,8476
Speedup	1,120067344	1,120037494	1,119807024	1,119691381	1,120170877
(%)	10,72%	10,72%	10,70%	10,69%	10,73%
BFS250	W32 OL1	W32 OL2	W32 OL4	W32 OL8	W32 OL16
PCD	19506,75	19505,7228	19506,3324	19502,5804	19504,2992
PCD Imp	17547,5084	17546,8188	17548,7576	17546,4556	17544,2448
Speedup	1,11165355	1,111638698	1,111550621	1,111482618	1,111720648
(%)	10,04%	10,04%	10,04%	10,03%	10,05%

Tabuľka C.3: PCD vs PCD Imp - prehľadané vrcholy pre test BFS 700

BFS700	W16 OL1	W16 OL2	W16 OL4	W16 OL8	W16 OL16
PCD	115288,4532	115291,2436	115293,5668	115283,4208	115288,9392
PCD Imp	99646,6396	99639,998	99632,536	99634,1216	99640,0704
Speedup	1,156972816	1,15707794	1,157187917	1,157067669	1,157053972
(%)	13,57%	13,58%	13,58%	13,57%	13,57%
BFS700	W32 OL1	W32 OL2	W32 OL4	W32 OL8	W32 OL16
PCD	66981,2424	66980,0768	66979,734	66982,3024	66982,0668
PCD Imp	59241,6292	59248,7812	59244,1092	59243,5332	59246,804
Speedup	1,130644841	1,130488686	1,13057205	1,130626396	1,130560001
(%)	11,55%	11,54%	11,55%	11,55%	11,55%

Tabuľka C.4: PCD vs PCD Imp - prehľadané vrcholy pre test BFS 950

BFS950	W16 OL1	W16 OL2	W16 OL4	W16 OL8	W16 OL16
PCD	171030,7712	171037,4376	171029,8776	171023,7532	171033,5692
PCD Imp	147996,6328	148010,3	147984,3516	147997,4772	148004,6488
Speedup	1,155639611	1,15557794	1,155729479	1,155585598	1,155595926
(%)	13,47%	13,46%	13,47%	13,46%	13,46%
BFS950	W32 OL1	W32 OL2	W32 OL4	W32 OL8	W32 OL16
PCD	96159,2756	96158,7308	96155,9648	96155,7188	96154,574
PCD Imp	85027,7484	85032,162	85033,6456	85024,4268	85027,2544
Speedup	1,130916406	1,130851298	1,13079904	1,130918754	1,130867681
(%)	11,58%	11,57%	11,57%	11,58%	11,57%

Tabuľka C.5: PCD vs PCD Imp - namerané časy pre test RND

RND100	W16 OL1	W16 OL2	W16 OL4	W16 OL8	W16 OL16
PCD	10,94496177	10,91177431	10,90933098	10,90008141	10,93913275
PCD Imp	9,709617012	9,686639839	9,710327788	9,695890022	9,726825235
Speedup	1,127228989	1,126476724	1,123477108	1,124196065	1,124635478
(%)	11,29%	11,23%	10,99%	11,05%	11,08%
RND100	W32 OL1	W32 OL2	W32 OL4	W32 OL8	W32 OL16
PCD	7,328404786	7,305868035	7,312651527	7,303524781	7,327211583
PCD Imp	6,786708659	6,766647948	6,784793994	6,768261831	6,789531425
Speedup	1,079817206	1,079687918	1,077800083	1,079084256	1,079192528
(%)	7,39%	7,38%	7,22%	7,33%	7,34%

Tabuľka C.6: PCD vs PCD Imp - namerané časy pre test BFS 250

BFS250	W16 OL1	W16 OL2	W16 OL4	W16 OL8	W16 OL16
PCD	9,736057128	9,766360845	9,765292754	9,770886661	9,78284547
PCD Imp	8,972674576	8,978025322	8,966779445	8,972118354	8,973893073
Speedup	1,085078596	1,087807229	1,089052409	1,089027839	1,09014509
(%)	7,84%	8,07%	8,18%	8,17%	8,27%
BFS250	W32 OL1	W32 OL2	W32 OL4	W32 OL8	W32 OL16
PCD	6,114246214	6,115216287	6,111831078	6,096530826	6,120681017
PCD Imp	5,719002908	5,689061746	5,70971134	5,707500347	5,710818897
Speedup	1,069110527	1,074907702	1,070427332	1,068161271	1,071769413
(%)	6,46%	6,97%	6,58%	6,38%	6,70%

Tabuľka C.7: PCD vs PCD Imp - namerané časy pre test BFS 700

BFS700	W16 OL1	W16 OL2	W16 OL4	W16 OL8	W16 OL16
PCD	35,52503923	35,4964511	35,54259132	35,48258655	35,57143077
PCD Imp	31,64698188	31,66059727	31,54304863	31,71366892	31,73629387
Speedup	1,122541143	1,121155447	1,126796326	1,118842056	1,12084388
(%)	10,92%	10,81%	11,25%	10,62%	10,78%
BFS700	W32 OL1	W32 OL2	W32 OL4	W32 OL8	W32 OL16
PCD	19,79863497	19,7802532	19,78705253	19,77680499	19,82775477
PCD Imp	18,2360159	18,29929974	18,18171962	18,24998413	18,28815846
Speedup	1,085688622	1,080929515	1,088293789	1,08366149	1,08418542
(%)	7,89%	7,49%	8,11%	7,72%	7,76%

Tabuľka C.8: PCD vs PCD Imp - namerané časy pre test BFS 950

BFS950	W16 OL1	W16 OL2	W16 OL4	W16 OL8	W16 OL16
PCD	52,34251021	52,36243428	52,26000939	52,30339712	52,38930806
PCD Imp	46,77306906	46,90258128	46,75153788	46,80215352	46,82393527
Speedup	1,119073673	1,116408369	1,117824391	1,117542531	1,118857434
(%)	10,64%	10,43%	10,54%	10,52%	10,62%
BFS950	W32 OL1	W32 OL2	W32 OL4	W32 OL8	W32 OL16
PCD	28,19893237	28,22296965	28,15197077	28,1378549	28,2039086
PCD Imp	25,90500041	25,98269477	25,84233978	25,85871395	25,8899312
Speedup	1,088551705	1,086221807	1,089373912	1,088138217	1,089377503
(%)	8,13%	7,94%	8,20%	8,10%	8,20%

Tabuľka C.9: PCD vs Waypoint A* - namerané časy pre test BFS 950

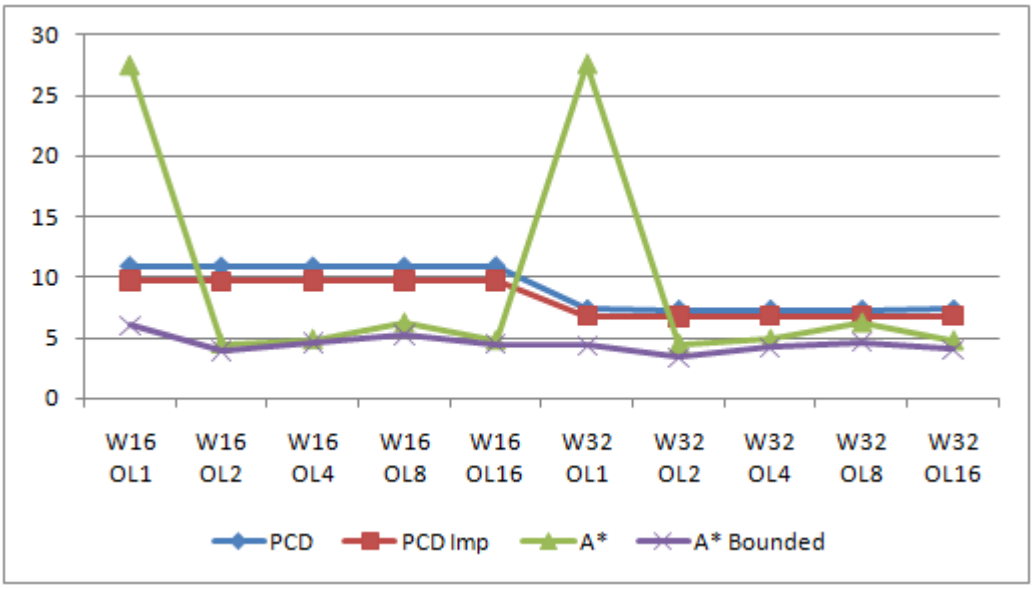
BFS950	W4 OL1	W8 OL1	W16 OL1	W32 OL1
PCD	404,1238114	306,46917	204,0950606	130,4394415
Waypoint A*	3653,3319	1825,589696	712,5924799	238,2753451
Speedup	9,04013027	5,95684615	3,491473422	1,826712399
(%)	88,94%	83,21%	71,36%	45,26%

Tabuľka C.10: PCD vs Waypoint A* - prehľadané vrcholy pre test BFS 950

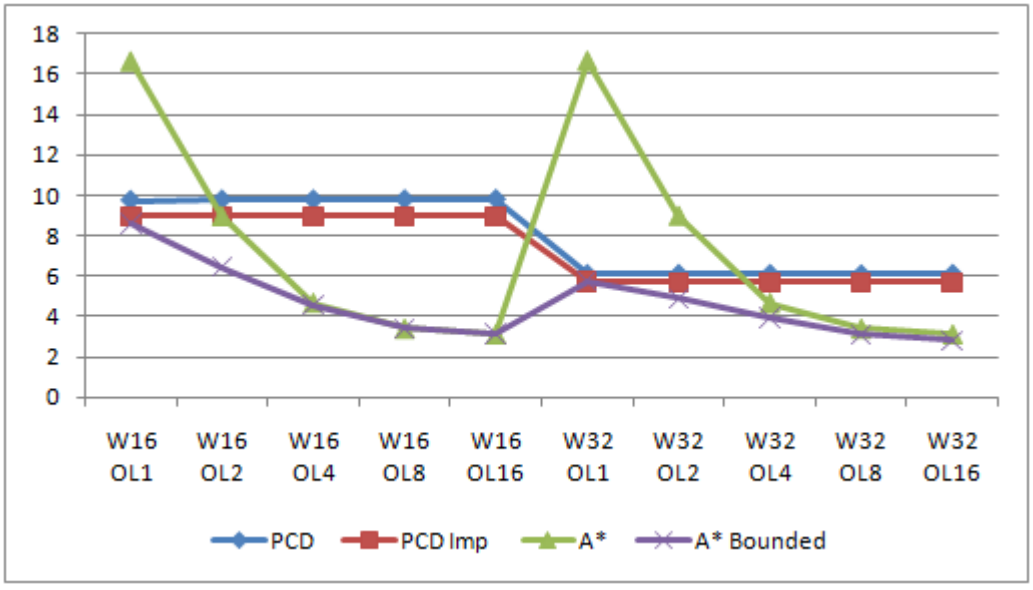
BFS950	W4 OL1	W8 OL1	W16 OL1	W32 OL1
PCD	701303,992	541677,236	370080,428	243494,816
Waypoint A*	356571,748	258334,276	166748,752	89979,728
Speedup	1,966796293	2,096807456	2,219389492	2,706107491
(%)	49,16%	52,31%	54,94%	63,05%

Tabuľka C.11: PCD vs Landmark A* - namerané časy pre test FFL

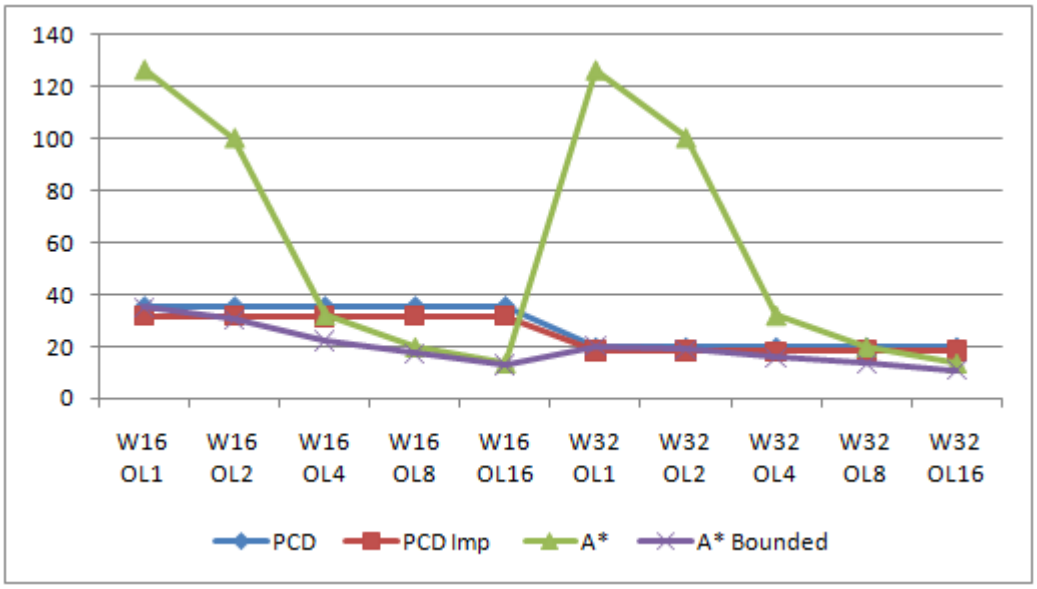
FFL	W32 OL16	W32 OL8	W32 OL4	W32 OL2	W32 OL1
PCD	28,28727073	28,43710856	28,39750636	28,29293746	28,35032955
PCD Imp	25,67636619	25,67685884	25,56565606	25,5913516	25,622681
A*	54,12181701	94,52544323	141,0943215	189,7472291	239,954881
A* Bounded	24,93371882	27,38155386	29,00449878	31,20089574	34,12587628
Speedup	2,107845659	3,681347622	5,518900873	7,414505966	9,36494042
(%)	52,56%	72,84%	81,88%	86,51%	89,32%



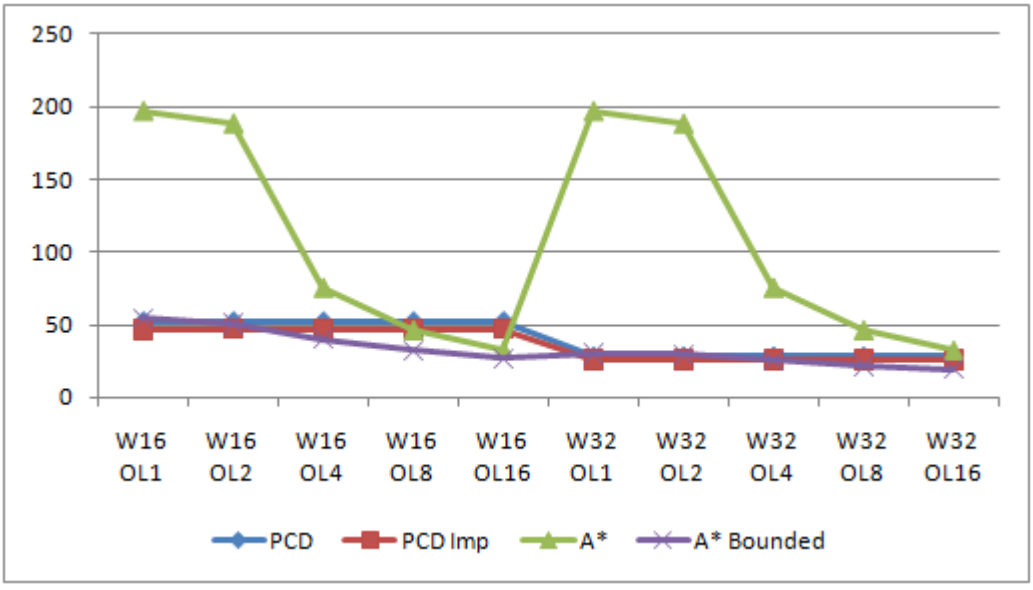
Obr. C.1: Porovnanie časov algoritmov v teste RND 100



Obr. C.2: Porovnanie časov algoritmov v teste BFS 250



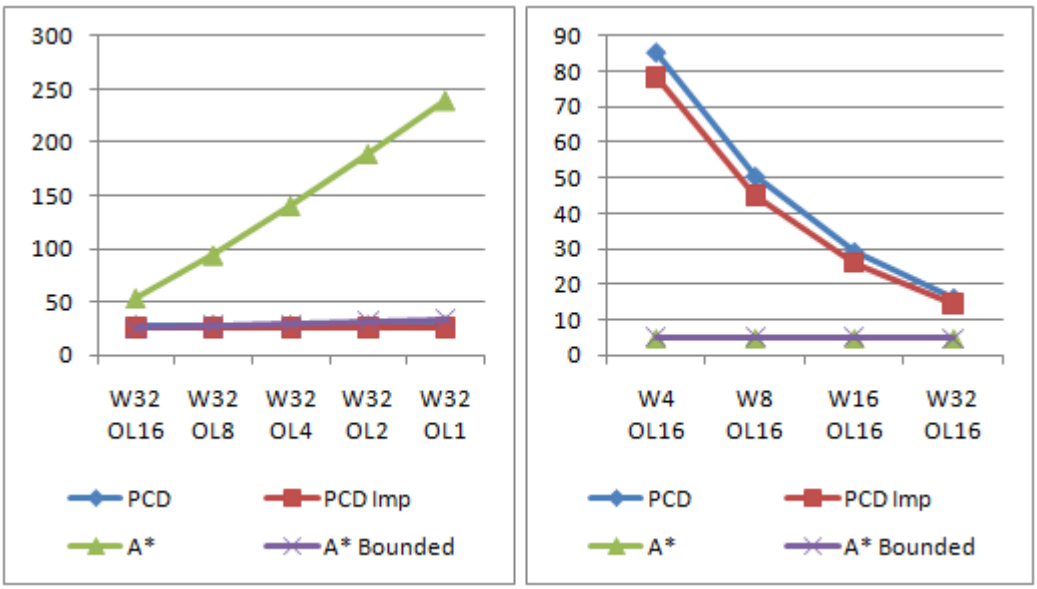
Obr. C.3: Porovnanie časov algoritmov v teste BFS 700



Obr. C.4: Porovnanie časov algoritmov v teste BFS 950

Tabuľka C.12: PCD vs Landmark A* - namerané časy pre test CTL

CTL	W4 OL16	W8 OL16	W16 OL16	W32 OL16
PCD	85,36455819	50,42384124	29,16665311	16,17939104
PCD Imp	78,37651162	44,93730212	26,07807429	14,55278289
A*	4,76471112	4,77683728	4,76705496	4,766156472
A* Bounded	5,158124453	5,146466815	5,052624632	4,780434304
Speedup	15,1947694	8,731680147	5,161292633	3,044238654
(%)	93,42%	88,55%	80,63%	67,15%



Obr. C.5: Časy algoritmov v testoch FFL (vľavo) a CTL (vpravo)

Literatúra

[AVG04] Chris Harrelson Andrew V. Goldberg. *Computing the Shortest Path: A* Search Meets Graph Theory. Technical Report MSR-TR-2004-24*, 2004.

[JM06] Domagoj Matijevic Jens Maue, Peter Sanders. *Goal Directed Shortest Path Queries Using Precomputed Cluster Distances*, volume 4007. 2006. `http://www.mpi-inf.mpg.de/~jensmaue/MaueSandersMatijevic2006.pdf`.

[Mau06] Jens Maue. *A Goal-Directed Shortest Path Algorithm Using Precomputed Cluster Distances*. 2006. `http://www.mpi-inf.mpg.de/~jensmaue/Maue2006.pdf`.

[MLF87] Robert E. Tarjan Michael L. Fredman. *Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM*, 34(3):596 – 615, 1987.

[Pat07] Amit Patel. Amit's notes about path-finding. 2007.

[Ďur] Pavol Ďuriš. *Tvorba efektívnyvh algoritmov (materíaly k prednáške)*.