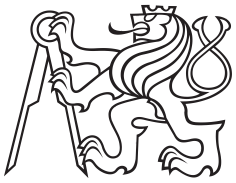
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra řídící techniky

Aplikace rychlých plánovacích algoritmů v úloze prohledávání neznámého prostoru

Václav Zelený

Vedoucí: RNDr. Miroslav Kulich, Ph.D.
Obor: Kybernetika a robotika
Studijní program: Systémy a řízení
Únor 2017

Poděkování

Na tom to místě bych rád poděkoval RNDr. Miroslavu Kulichovi, Ph.D., za konzultace obsahu této práce a za odborné vedení při zpracování implementační části. Dále za pomoc při hledání podkladů k seznamování se problematikou v této oblasti.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 10. února 2017

Abstrakt

Tato bakalářská práce se zabývá algoritmy pro plánování tras a jejich aplikací pro rychlé hledání cest na mřížkových mapách.
Tato práce má dvě části. V první části se popisují základní algoritmy pro hledání nejkratších cest na obecných grafech i na mřížkových mapách. Druhá část se zabývá implementací a testováním konkretního algoritmu pro rychlé hledání cest na mřížkové mapě.

Klíčová slova: mřížková mapa, plánovací algoritmus, Dijkstrův algorimus, A*, Grafy podcílů, Jump Point Search, Ohraničení cílů, ROS

Vedoucí: RNDr. Miroslav Kulich, Ph.D.
Jugoslávských partyzánů,
Praha

Abstract

This bachelor thesis deals with route planning algorithms and their applications for fast search of paths on grid maps.
The thesis is divided into two parts. The first part describes the basic algorithms for finding the shortest paths in the general graphs and on the grid maps. The second part deals with the implementation and testing of a specific algorithm for fast search of paths on a grid map.

Keywords: grid map, planning algorithm, Dijkstra algorithm, A*, Subgoal Graphs, Jump Point Search, Goal Bounds, ROS

Title translation: Application of fast planning algorithms in the exploration task

Obsah

Obrázky

Kapitola 1
Úvod

Jeden z problémů řešený v robotice je, jak co nejoptimálněji prohledat prostor, který je neznámý. Roboti jsou pro tento účel vybaveni různými senzory, například laserovým skenerem nebo kamerami, pomocí nichž si vytváří virtuální mapu prostředí. Na základě této mapy se posléze rozhodují, jakým směrem prozkoumávat neznámý prostor tak, aby to bylo co nejvíce optimální. Požadavek na optimálnost chodu robota může například být, aby při prohledávání neznámého prostoru urazil co nejkratší cestu. Tento problém se řeší pomocí různých algoritmů, jako je algoritmus pro řešení problému obchodního cestujícího.

V každém případě všechny tyto rozhodovací algoritmy potřebují znát nejkratší cestu mezi určitými body mapy. Pro tento účel tedy musí implementovat plánovací algoritmus, který tento problém vyřeší.

Robot potřebuje plánovat trasu po každé aktualizaci mapy, proto na hledání nejkratších cest mezi vybranými body mapy není moc času. Mapa prostředí, kterou robot prohledává, může časem narůst do velkých rozměrů. Algoritmus si tedy musí poradit se zpracováním velkého množství bodů ve velice krátkém čase.

Cílem této práce je seznámit se s plánovacími algoritmy a z nich vybrat nejrychlejší možný algoritmus pro hledání nejkratších cest na mřížkových mapách. Následně najít vhodnou implementaci tohoto algoritmu a tu upravit pro softwarové simulační prostředí používané na výzkumném pracovišti vedoucího této bakalářské práce.

Dále implementovat výpočetní uzel do ROS tak, aby implementovaný uzel využíval tento algoritmus k vypočtu matice vzdáleností, pro vybrané uzly z mapových podkladů získávaných od uzlu s názvem *Gmapping*, který je poskytuje ve formě mřížkové mapy prostředí.

Poslední částí práce je experimentální ověření implementovaného plánovacího algoritmu v simulovaném prostředí a porovnání rychlostí výpočtu nejkratších cest s jinými plánovacími algoritmy.

Kapitola 2
Plánovací algoritmy

Plánovací algoritmy jsou algoritmy sloužící k naplánování nejkratší cesty. Dají se rozdělit do tří kategorií: které hledají cestu mezi dvěma vrcholy grafu, ty které hledají nejkratší cesty z jednoho vrcholu do všech ostatních vrcholů a na ty které hledají nejkratší cesty mezi všemi vrcholy.

Obecně tyto algoritmy pracují s obecnými grafy, ve kterých hledají nejkratší sled hran mezi vrcholy. Většina algoritmů však využívá nějakou specifickou vlastnost grafu a tím snižuje časovou nebo paměťovou náročnost algoritmu. Nejvšeobecnější algoritmus je Bellmanův-Fordův-Mooreův algoritmus. Ten umí najít cestu z jednoho uzlu do všech ostatních vrcholů na obecných grafech i se zápornými hodnotami hran. Cenou za to je vyšší časová složitost algoritmu.

Většina grafů, popisující nějakou reálnou mapu, však hrany záporné délky neobsahuje. V tomto případě se dá použít Dijkstrův algoritmus, který umí hledat nejkratší cesty z jednoho vrcholu do všech ostatních vrcholů na obecných grafech s nezápornou délkou hran mezi vrcholy.

Dalším velice požívaným algoritmem je algoritmus A*, který umí najít cestu mezi dvěma vrcholy na obecných grafech s nezápornou délkou hrany. K zrychlení svého běhu používá heuristikou funkci, která odhaduje vzdálenost do cíle a díky tomu upřednostňuje vrcholy blíže k cíli.

Mnoho jiných algoritmů využívá dalších vlastnosti grafu pro zlepšení své časové náročnosti, například že je graf reprezentovaný čtvercovou mřížkou, délky hran jsou násobky nějaké dané hodnoty nebo se po grafu dá pohybovat jen specifickým způsobem a podobně.[1]

2.1 Specifika plánování na mřížkách

Mřížka je speciální druh grafu, kde jednotlivé vrcholy tvoří 2D sít, kde každý vrchol je spojen se všemi svými sousedy a kde poměr délky hran pro horizontální a vertikální pohyb k pohybu diagonálnímu je jedna ku odmocnině

ze dvou, pokud je diagonální pohyb v daném případě povolený. V plánovacím algoritmu se dají využít některé její vlastnosti pro zrychlení běhu algoritmu.

Mřížky mají tu výhodu, že se jimi dobře reprezentují mapy, které se dají lehce vytvořit, například laserovým skenerem robota, a snadno tak reprezentují reálné prostředí.

Další výhodou je, že lze mřížku v počítači uložit jako dvourozměrné pole, kde v každé buňce tohoto pole je uložena informace, zda je tam překážka nebo je buňka pro průchod robota volná. A u každé buňky se tedy nemusí ukládat další informace, zda existuje hrana do vedlejší buňky, protože je jasné, jaké buňky spolu sousedí.

Základní vlastností mřížky je symetričnost. V grafu, který je sestaven nad mřížkovou mapou, většinou existuje velké množství cest, které se mezi sebou dají zaměnit i když procházejí přes jiné vrcholy, protože délka těchto cest je stejná. Při hledání nejkratší cesty se této symetričnosti dá využít. Některé hrany se nemusejí expandovat a některé buňky se nemusejí vůbec přidávat do množiny pro pozdější zpracování expandovaných uzlů, ale mohou se zpracovat přímo po objevení. Tomuto postupu expandování uzlů se říká skoky po mřížce. Na těchto principech pracuje algoritmus Jump Point Search. Jiné algoritmy využívají toho, že nejkratší cesty musejí jít přes konkrétní buňky, jako jsou buňky na hraně překážek. Toho zase využívá algoritmus Grafů podcílů.

Další vlastností mřížek je, že když se expanduje směrem k cíli, nemusí se expandované buňky přidávat do množiny nezpracovaných buněk, která je většinou implementována pomalejší prioritní frontou, ale může se použít rychlý zásobník. Pokud se v algoritmu A* jako heuristická funkce použije eukleidovská vzdálenost nebo octile heuristika, pak navštíví tento algoritmus každou buňku maximálně jednou, protože to jsou funkce na mřížce monotónní.

Nevýhodu plánovaní na mřížkách je, že pro reprezentaci mapy je potřeba hodně velké množství vrcholů (buněk) a to zpomaluje obecnější algoritmy, které neumějí využít vlastnosti mřížek.

Kapitola 3
Algoritmy

3.1 Dijkstrův algoritmus

Dijkstrův algoritmus je algoritmus pro hledání nejkratších cest v grafech s nezápornou délkou hran grafu. Je nejrychlejší známý algoritmus pro nalezení všech nejkratších cest v grafu ze zadaného vrcholu. Výhodou je i to, že hledá cestu do všech vrcholů z jednoho počátečního vrcholu a ne jen cestu z počátečního vrcholu do cílového vrcholu jako například algoritmus A*.

Jde použít nejen na grafy reprezentované mřížkovou mapu, ale i na obecné grafy.

Dijkstrův algoritmus pracuje takto. Mějme graf G, ve kterém hledáme nejkratší cestu. Množina vrcholů V, která obsahuje všechny vrcholy grafu G, a množinu hran E mezi vrcholy z množiny V. Pro každý vrchol v si algoritmus pamatuje délku nejkratší cesty $d(v)$, která vede z počátečního vrcholu až do vrcholu v. Na začátku algoritmu je pro všechny vrcholy délka cesty z počátku $d(v)$ rovna nekonečnu, kromě počátečního vrcholu pro který je hodnota nejkratší cesty $d(v)$ rovna nule.

Dále se v tomto algoritmu požívají dvě pomocné množiny. Množina otevřených vrchů O (open list), ve které jsou uchovávané všechny nezpracované vrcholy grafu, a množina uzavřených vrcholů C (close list), ve které jsou již zpracované vrcholy.

Z množiny nezpracovaných vrcholů O se vezme vrchol v s nejmenší hodnotou délky cesty od počátečního vrcholu $d(v)$ a přidá se do množiny již zpracovaných vrcholů C. Pro všechny vrcholy u, do kterých vede hrana z tohoto vrcholu v, se nastaví nová hodnota nejkratší cesty z počátečního vrcholu $d(u) = d(v) + l(v, u)$, pokud je $d(u) > d(v) + l(v, u)$, kde $l(v, u)$ je délka hrany z vrcholu v do vrcholu u. Tedy když je nalezena kratší cesta do tohoto vrcholu.

Pokud chceme znát i přes jaké vrcholy grafu vede nejkratší cesta, musí se u každého vrcholu u ukládat informace o tom, z jakého vrcholu v je nalezena nejkratší cesta. Tedy pokud platí podmínka $d(u) > d(v) + l(v, u)$, tak u

vrcholu u změníme informaci, že zatím nejkratší nalezená cesta z počátečního vrcholu vede přes vrchol v.

Algoritmus končí, když je množina nezpracovaných vrcholů O prázdná nebo případně, že jsme nalezli cestu do všech požadovaných vrcholů, pokud není potřeba znát nejkratší cesty do všech vrcholů grafu nebo jen jejich délek.

Po skončení běhu Dijkstrova algoritmu je pro každý vrchol v grafu známa délka nejkratší cesty z počátečního vrcholu $d(v)$, případně jen pro požadované vrcholy.

Když je potřeba najít nejkratší cestu z nějakého vrcholu do počátečního vrcholu, stačí jít po vrcholech, ze kterých byla nalezena nejkratší cesta do daného vrcholu, dokud se nedojde do počátečního vrcholu.

Složitost tohoto algoritmu záleží na způsobu reprezentace množiny nezpracovaných vrcholů O. Pokud je pro množinu nezpracovaných vrcholů O použito pole, tak kvůli sekvenčnímu prohledávání bude asymptotická složitost $O(|V|^2)$, kde je V množina vrcholů grafu. Při reprezentaci množiny nezpracovaných vrcholů O pomocí binární haldy bude asymptotická složitost $O((|E|+|V|)\log|V|)$ a při použití Fibonacciho haldy bude asymptotická složitost dokonce $O(|E|+|V|\log|V|)$, kde množina E je množina hran grafu.[1]

3.2 A*

A* je algoritmus pro nalezení nejkratší cesty v grafech s nezápornou délkou hran grafu. Je podobný Dijkstrovu algoritmu, ale hledá cestu z počátečního vrcholu do cílového vrcholu, přičemž využívá navíc heuristickou funkci.

A* pracuje stejně jako Dijkstrův algoritmus. Jen se pro výběr dalšího vrcholu v z množiny nezpracovaných vrcholů O nepoužívá vzdálenost z počátečního vrcholu $d(v)$, ale hodnotící funkce $f(v)$, která se skládá ze dvou funkcí: z funkce $d(v)$ a z funkce $h(v)$, kde funkce $d(v)$ je délka cesty z počátečního vrcholu do vrcholu v a funkce $h(v)$ je heuristická funkce neboli odhad vzdálenosti z aktuálního vrcholu v do cílového vrcholu.

Aby heuristická funkce $h(v)$ byla přípustná, musí platit, že $h(v) \leq d(v)$, kde $d(v)$ je skutečná vzdálenost z vrcholu v do cílového vrcholu. V případě že tato podmínka nebude platit, nemusí algoritmus A* najít nejkratší cestu z počátečního vrcholu do cílového vrcholu.

Pokud navíc pro heuristikou funkci platí, že $h(x) \leq l(x,y) + h(y)$, kde x a y jsou dva sousední vrcholy a $l(x,y)$ je délka hrany mezi těmito dvěma sousedními vrcholy, tak pak je heuristická funkce $h(v)$ monotónní a každý vrchol bude při běhu algoritmu A* navštíven právě jednou. Tato podmínka u grafů, které reprezentující mapu reálných vzdáleností, platí vždy.

Jako heuristikou funkci se dá požít jakákoliv funkce, která splňuje podmínku přípustnosti. Nejčastěji se používají některé z následujících funkcí.

Eukleidovská vzdálenost je heuristická funkce $h(v) = \sqrt{(x_g - x_v)^2 + (y_g - x_v)^2}$, kde x_g a y_g jsou souřadnice cílového vrcholu a x_v a y_v jsou souřadnice vrcholu v. Tato heuristická funkce se dá požít na většinu grafů reprezentující nějakou mapu, ale v některých případech se vyplatí požít některou z následujících funkcí, protože jejich výpočet je rychlejší a nemusí počítat funkci druhé odmocniny.

Manhattanská vzdálenost je heuristická funkce $h(v) = abs(x_g - x_v) + abs(y_g - x_v)$, kde x_g a y_g jsou souřadnice cílového vrcholu a x_v, y_v jsou souřadnice vrcholu v a funkce abs je absolutní hodnota. Tato heuristická funkce se používá na mapách reprezentovaný čtvercovou mřížkou, kde nejsou povoleny diagonální cesty. A je povoleno se po ní pohybovat jen vertikální nebo horizontálně.

Při použití na mřížce, kde jsou povolené i diagonální cesty, se jako heuristická funkce používá tzv. octile heuristika $h(v) = abs(x_g - x_v) + abs(y_g - x_v) + (\sqrt{2} - 2) \cdot min(x_g - x_v, y_g - x_v)$, x_g a y_g jsou souřadnice cílového vrcholu, x_v a y_v jsou souřadnice vrcholu v, funkce abs je absolutní hodnota a funkce min je minimum z daných hodnot.

Algoritmus končí ve chvíli, kdy hodnota cílového vrcholu je menší než hodnota jakéhokoliv vrcholu v množině otevřených vrcholů nebo když je množina otevřených vrcholů prázdná.

Složitost tohoto algoritmu je závislá na použité heuristické funkci. V nejhorším případě bude asymptotická časová složitost stejná jako u Dijkstrova algoritmu.[1, 2]

3.3 Grafy podcílů

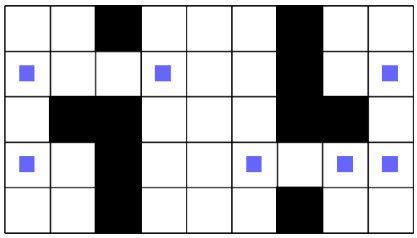
Grafy podcílů neboli Subgoal Graphs je algoritmus pro snížení počtu vrcholů při hledání nejkratší cesty. Dá se použít pouze na mapách reprezentovaných osmisměrnou mřížkou, tedy na grafech, které reprezentují mřížkovou mapu. Z každé buňky se může pohybovat osmi směry. Nedá se tedy použít na obecné grafy.

Nejprve si musíme definovat povolené tahy na osmisměrné mřížce, které byly použity pro tento algoritmus a další algoritmy pracující na mřížkách. Povolené tahy jsou posun o jednu buňku vertikálně, horizontálně nebo diagonálně, s výjimkou pro diagonální tah. Pokud není alespoň jeden z horizontálního nebo vertikálního sousedního tahu přípustný kvůli blokované buňce, tak i tento diagonální tah poté není přípustný.

Pro samotné hledání cesty mezi dvěma vrcholy se používá algoritmus A* s oktile heuristickou funkcí, ale před samotným hledáním cesty se mapa předzpracuje a až na takto předzpracovanou mapu se použije algoritmus A*.

Tento základní algoritmus se dělí na dva další algoritmy. Každý z nich předzpracovává mapy jiným způsobem: jednoduché grafy podcílů (simple subgoal graphs) a dvouúrovňové grafy podcílů (two-lavel subgoal graphs).

Oba tyto algoritmy pracují s grafy podcílů. Grafy podcílů obsahují určité buňky z původní mapy nazývané podcíle. To jsou buňky, které leží na rohu překážky, viz obrázek 3.1. [3, 4, 5]



Obrázek 3.1: Na hranách překážek jsou modře vyznačeny podcíle tohoto grafu

Pro vysvětlení jednotlivých algoritmů je potřeba definovat dva druhy spojení dvou různých buněk:

Dvě buňky jsou navzájem *h-dosažitelné* právě tehdy, když je délka cesty mezi těmito buňkami rovna hodnotě heuristické funkce mezi těmito buňkami.

A dvě buňky jsou navzájem *přímo-h-dosažitelné* právě tehdy, když žádná z nejkratších cest mezi těmito buňkami nevede přes nějakou buňku z množiny podcílů.

3.3.1 Jednoduché grafy podcílů

Tento algoritmus využívá jen jednu množinu podcílů. Nejprve se vypočítá množinu podcílů, tedy naleznou se všechny buňky, které se nacházejí na hraně překážek. Poté si algoritmus vytvoří graf a jako množinu vrcholů použije nalezenou množinu podcílů.

Mezi každou dvojici podcílů přidá hranu, pokud jsou tyto vrcholy přímo-h-dosažitelné. Takto vytvořený graf se použije pro hledání nejkratší cesty.

Algoritmus hledání nejkratší cesty potom probíhá ve třech krocích. V prvním kroku se nejprve otestuje jestli počáteční buňka a cílová buňka nejsou přímo-h-viditelné. Pokud jsou přímo-h-viditelné, tak se pro nalezení cesty použije algoritmus A* přímo na osmisměrnou mřížku a hledání cesty skončí. Pokud nejsou přímo-h-viditelné, tak se v druhém kroku počáteční buňka a cílová buňka přidá do množiny vrcholů grafu podcílů a doplní se všechny hrany mezi h-dosažitelné vrcholy.

V posledním kroku se na výsledný graf podcílů použije algoritmus A*, který najde nejkratší cestu mezi počátečním a cílovým vrcholem.

Čas nutný pro hledání cesty je kratší než při použití samotného algoritmu A*, protože množina podcílů je výrazně menší než množina všech uzlů grafu, za cenu předzpracování mapy. Zrychlení hledání nejkratší trasy tedy nastane až při vyhledávání více cest na stejné mapě, protože graf podcílů se nemusí pokaždé počítat znovu, pokud se nezmění mapa. A tak se čas nutný pro jeho výpočet rozloží.[3, 4, 5]

3.3.2 Dvouúrovňové grafy podcílů

Tento algoritmus využívá na rozdíl od předchozího dvě množiny podcílů. Jedna množina se nazývá množina lokálních podcílů a druhá množina se nazývá množina globálních podcílů.

Algoritmus nejdříve nalezne všechny podcíle, které se nacházejí na mapě, a vytvoří si z nich graf. Mezi každou dvojici podcílů v tomto grafu přidá hranu, pokud jsou tyto podcíle přímo-h-viditelné, jako v předchozím algoritmu. Vezmou se a všechny se označí jako globální podcíle. Poté se postupně jeden po druhém zpracovávají a když každá z dvojic jejich sousedů splňuje alespoň jednu z následujících dvou podmínek, tak se přidají do množiny lokálních podcílů.

První podmínkou je, že existuje cesta mezi těmito dvěma vrcholy jdoucí pouze přes globální podcíle, která neobsahuje právě zpracovávaný vrchol a která není delší než originální cesta.

Druhou podmínkou je, že jsou tyto dva podcíle navzájem h-dosažitelné. Pokud je právě zpracovávaný vrchol vyhodnocen jako lokální, tak se mezi sousedy, které byly vyhodnoceny jako h-dosažitelné, přidá hrana o délce hodnoty heuristické funkce mezi těmito vrcholy. Uložíme si u této hrany poznámku, přes který vrchol vedla původní cesta, pro pozdější vypsání nejkratší cesty.

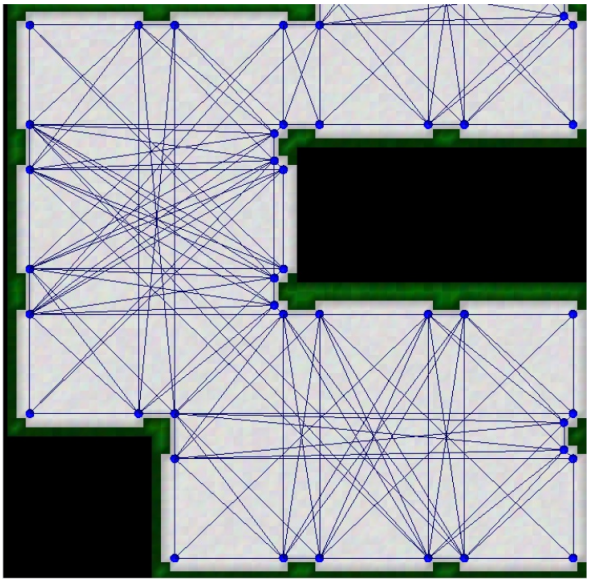
Výsledné množiny globálních podcílů a lokálních podcílů se mohou lišit podle toho, v jakém pořadí se zpracovávají a přesouvají vrcholy z původní množiny globálních podcílů do množiny lokálních podcílů. Takto vypočítané množiny se použijí pro hledání nejkratší cesty.

Algoritmus hledání nejkratší cesty potom probíhá opět ve třech krocích. V prvním kroku se testuje, zda počáteční a cílový vrcholy jsou přímo-h-dosažitelné. Pokud jsou přímo-h-viditelné, tak se pro nalezení cesty použije algoritmus A* přímo na osmisměrnou mřížku a hledání cesty skončí.

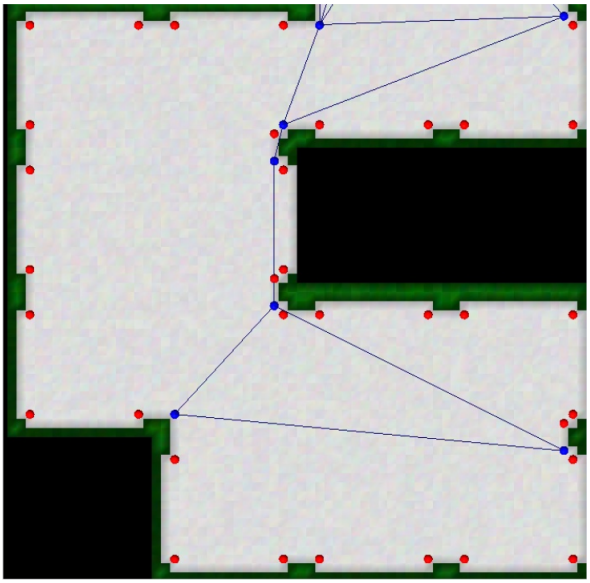
Pokud nejsou přímo-h-viditelné, tak se v druhém kroku přidá počáteční vrchol, cílový vrchol a všechny lokální podcíle, které jsou přímo-h-dosažitelné z počátečního nebo cílového vrcholu, dočasně do množiny globálních podcílů.

V posledním kroku se na výsledný graf globálních podcílů použije algoritmus A*, který najde nejkratší cestu mezi počátečním a cílovým vrcholem.

Rychlost hledání cesty je lepší než při použití samotného algoritmu A* nebo jednoduchého grafu podcílů, protože množina globálních podcílů je výrazně menší než množina všech vrcholů grafu nebo všech podcílů, za cenu předzpracování mapy. Zrychlení hledání nejkratší trasy nastane tedy až při hledání více cest na stejné mapě, protože grafy podcílů se nemusí pokaždé počítat znovu, pokud se nezmění mapa. [3, 4, 5]



Obrázek 3.2: Graf lokálních podcílů [3]



Obrázek 3.3: Graf globálních podcílů [3]

3.4 Jump Point Search

Jump Point Search algoritmus se jako algoritmus Grafy podcílů nedá použít na obecné grafy, ale pracuje pouze na osmisměrné mřížkové mapě, kde délka cesty mezi buňkami odpovídá reálné vzdálenosti. Povolené tahy jsou posun o jednu buňku vertikálně, horizontálně nebo diagonálně, s výjimkou pro

diagonální tah. Pokud není alespoň jeden z horizontálního nebo vertikálního sousedního tahu přípustný kvůli blokované buňce, tak i tento diagonální tah poté není přípustný. Existuje také verze tohoto algoritmu, která povoluje diagonální tah jen, pokud jsou oba tahy přípustné.

K hledání cesty se používá algoritmus A* s octile heuristikou s kombinací prořezávaní možných cest a skoků po mřížce.

Problém použití algoritmu A* na mřížkových mapách je, že na nich existuje díky symetrii spousta permutací pro nejkratší cestu. Při procházení těchto stejně dlouhých cest se zvyšuje počet operací při manipulaci se zpracovávanými buňkami na množině otevřených buněk a množině uzavřených buněk.

Jump Point Search využívá vlastnosti mřížky a eliminuje tyto symetrie tak, že neexpanduje do všech sousedů každé buňky, a tím snižuje počet operací nutných k přidání a odebírání buněk z množiny otevřených buněk. Dále nepřidává expandovanou buňku do množiny otevřených buněk, pokud to není nutné, ale provede skok až na místo buňky, která se do množiny otevřených buněk musí přidat.

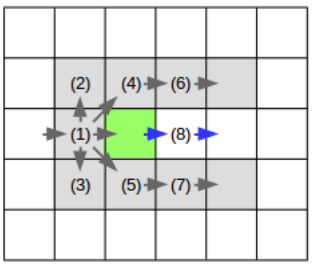
Jump Point Search provádí všechny optimalizace za běhu, a proto nepotřebuje provádět žádné předběžné zpracování a ani nezvyšuje nároky na paměť. K tomu používá již zmiňované dva postupy: prořezávání a skoky.

Pro vybírání buněk z množiny otevřených buněk algoritmus pracuje stejně jako A* s octile heuristikou. [6, 7, 8, 9]

Vertikální a horizontální skoky

Při horizontálním a vertikálním pohybu je na volné mřížce několik pravidel, které zjednoduší prohledávání. Pomocí nich se může expanze do některých buněk ignorovat, protože tyto buňky budou navštívené z jiných buněk. Dále se některé další buňky po expanzi nemusejí přidávat do množiny otevřených buněk a místo toho se provádí skoky až do buňky, která se již do množiny otevřených buněk musí přidat podle později popsaných pravidel.

Pro vysvětlení budeme předpokládat, že se do do aktuální buňky přišlo zleva, viz obrázek 3.4. Pro ostatní horizontální a vertikální směry platí obdobná pravidla.

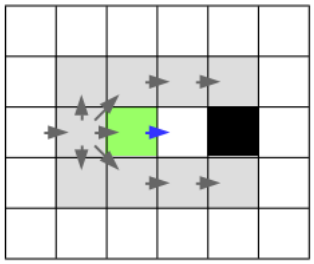


Obrázek 3.4: Horizontální (vertikální) skok

Při příchodu do buňky se může ignorovat buňka (1), ze které se přišlo, protože již byla navštívená. Pak se můžou ignorovat diagonální buňky $(2, 3)$, nad a pod buňkou ze které se přišlo do aktuálně zpracovávané, protože se na ně dá dostat kratší cestou právě přes ni. Dále se můžou ignorovat buňky nad a pod $(4, 5)$ aktuálně zpracovávanou buňkou, protože jsou optimálně dosažitelné právě přes buňku, přes kterou se na ní přišlo, s cenou cesty $\sqrt{2}$, ale přes aktuálně zpracovávanou buňku by byla cena cesty 2. Sousední buňky diagonálně vpravo nehoře a dole $(6, 7)$ jsou dosažitelné přes aktuálně zpracovávanou buňku, ale i přes buňky nad ní a pod ní. Protože cena bude stejná, můžou se tyto buňky také ignorovat. To jsou všechny buňky, které se mohou ignorovat při expanzi.

Tím pádem zbývá jediná sousední buňka (8), která se musí navštívit a to je buňka v přímém směru od buňky, ze které se přišlo. Tato buňka se hned nepřidá od množiny otevřených buněk, ale bude se opakovat skok vpravo dokud se nenarazí na překážku nebo buňku sousedící vertikálně s hranou.

Pokud se narazí na překážku, viz obrázek 3.5, může se celý tento skok bezpečně ignorovat, protože buňky nad a pod tímto celým přeskočeným řádkem budou zpracovány přes jiné buňky. Pokud se ale narazí na buňku sousedící vertikálně nebo diagonálně s hranou překážky, tak se musí tato buňka přidat do množiny otevřených buněk a skok zastavit.[6, 7, 8, 9]



Obrázek 3.5: Při naražení na překážku můžeme skok ignorovat

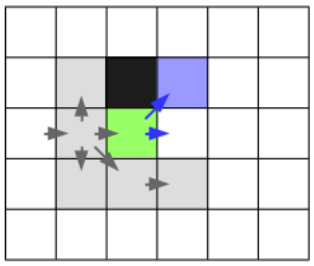
Když se bude tato buňka zpracovávat, musí se do množiny otevřených buněk přidat i sousední buňka za rohem, viz obrázek 3.6, pokud je volná i vertikální buňka. Této buňce je říká vynucený soused (forced neighbor), protože se musí přidat do množiny otevřených buněk a nemůže se ignorovat.

Diagonální skoky

Pro diagonální pohyb na volné mřížce existují také podobná zjednodušující pravidla.

Kvůli vysvětlení teď budeme předpokládat pohyb směrem vpravo nahoru, viz obrázek 3.7. Pro ostatní diagonální směry se použijí obdobná pravidla.

Při příchodu do buňky se může ignorovat buňka (1), ze které se přišlo, protože již byla navštívená. Dále se mohou ignorovat buňky vlevo a dole $(2, 3)$. Ty již byly navštívené z předchozí buňky (1) expanzí v horizontálním a



Obrázek 3.6: Nalezení hrany (zeleně) a vynucený soused (modře)

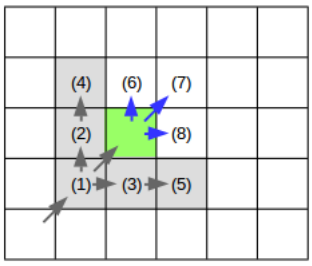
vertikálním směru. Také se mohou ignorovat buňky diagonálně vlevo nahoře (4) a vlevo dole (5), tyto buňky lze také efektivněji dosáhnout z buněk vlevo (2) a dole (3). Tím zbývají tři buňky k expanzi: nahoru (6), diagonálně (7) a vpravo (8). Tedy horizontální nebo vertikální směr a jeden směr diagonální.

Nejdříve se musí prozkoumat horizontální a vertikální směr. To se provede obdobným způsobem, jak bylo vysvětleno v odstavci vertikální a horizontální skoky. Pokud se při tom narazí na nějakou buňku sousedící s hranou překážky, tak se skok přeruší a aktuálně zpracovávaná buňka (zelená na obrázku 3.7) se přidá do množiny otevřených buněk a skok ukončíme.

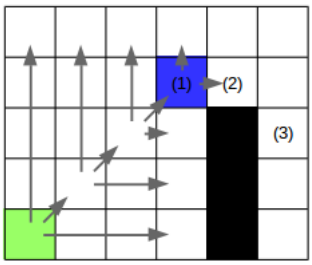
Když se na žádné hrany překážek nenarazí, expanduje se zbylým diagonálním směrem. Popřípadě se tento postup opakuje, dokud se nenarazí na překážku.

Na obrázku 3.8 je uveden příklad diagonálního skoku, který začne v zelené buňce a v buňce (1) je nalezna hrana (2). Na buňce (1) se tedy zastaví skok a buňka se přidá do množiny otevřených buněk.

Až bude tato buňka (2) vyjmuta z množiny otevřených buněk pro další zpracování, tak teprve v tento okamžik se buňka přidá do množiny otevřených buněk a při zpracování této buňky, bude nalezena buňka (3) za rohem překážky, protože je to její vynucený soused .[6, 7, 8, 9]



Obrázek 3.7: Diagonální skok

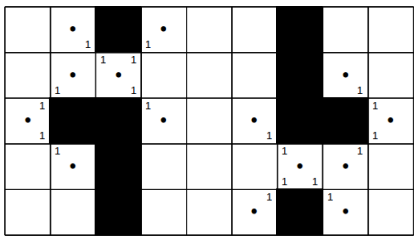


Obrázek 3.8: Příklad diagonálního skoku

3.4.1 Jump Point Search Plus

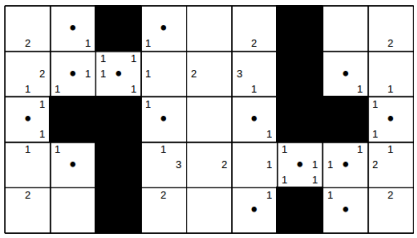
Jump Point Search Plus algoritmus pracuje na stejných grafech jako Jump Point Search, ale nepočítá prořezávaní a skoky za běhu, ale před samotným hledáním cesty provede předzpracování mapy.

Nejdříve se naleznou všechny primární buňky. To jsou buňky, přes které vedou cesty na vynucené sousedy, a skoky vedoucí na vynucené sousedy z primárních cesty se uloží. (Obrázek 3.9)



Obrázek 3.9: Primární buňky a skoky na vynucené sousedy

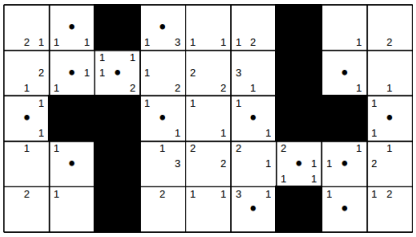
Poté se pro každou buňku spočítají maximální délky vertikálních a horizontálních skoků na primární buňky. (Obrázek 3.10)



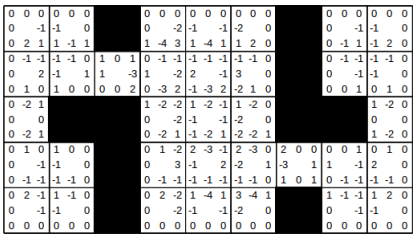
Obrázek 3.10: Přidání horizontálních a vertikálních skoků

Dále se pro každou buňku spočítají délky všech diagonálních skoků potřebných k dosažení vertikálního nebo horizontálního skoku.

Nakonec se spočítají skoky ke stěnám ve směrech, kde neexistuje skok po mřížce. Tyto hodnoty se ukládají jako záporná hodnota vzdálenosti, kvůli odlišení délky skoku ke stěně a skoku po mřížce.



Obrázek 3.11: Přidání diagonálních skoků



Obrázek 3.12: Přidání skoků ke stěnám

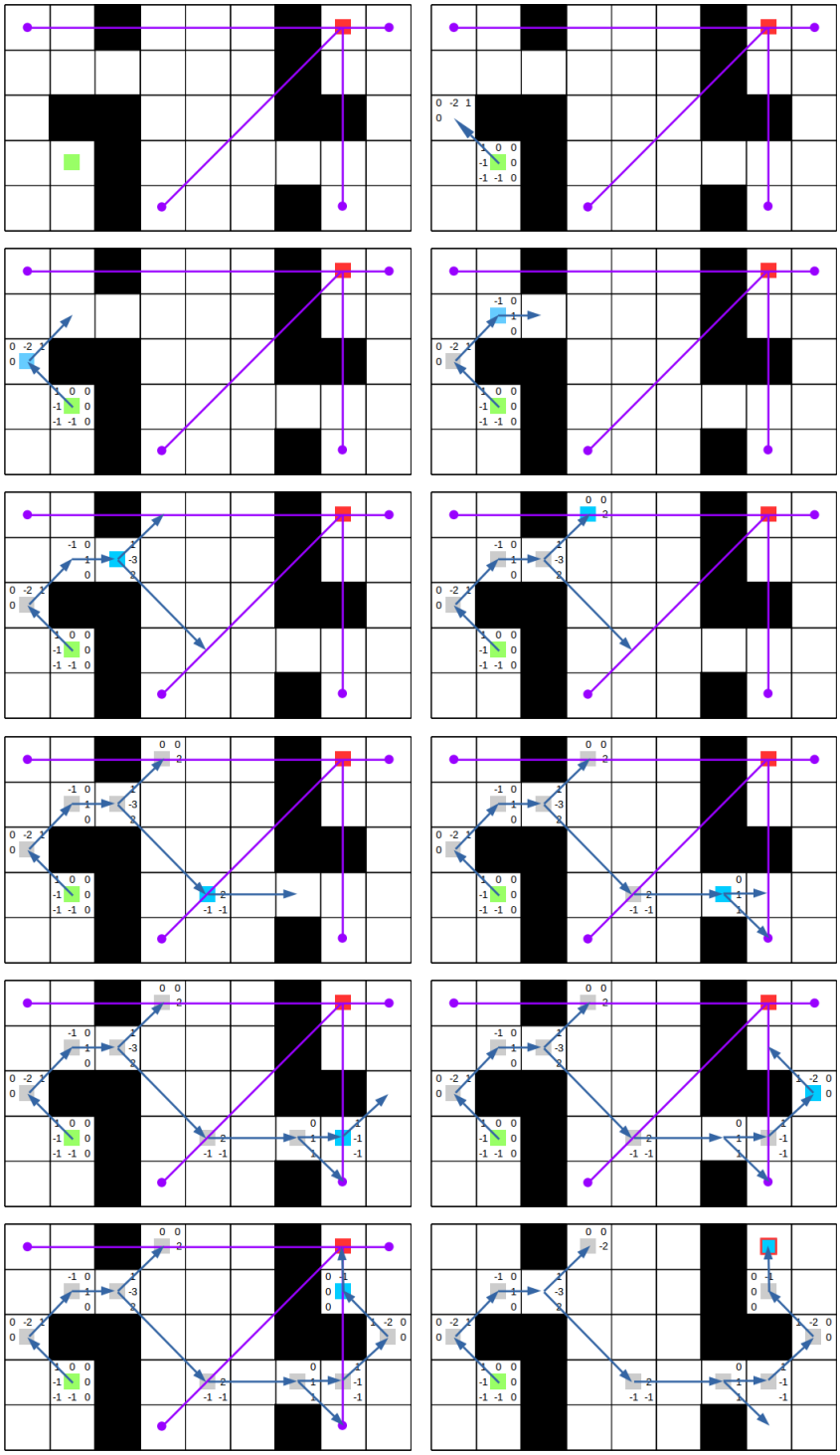
Při hledání cíle se použijí tyto přepočítané hodnoty k rychlým skokům, protože se nemusí procházet celá trasa a kontrolovat, jestli je tam překážka, ale rovnou se skočí o uložený počet buněk.

Před každým skokem se musí zkontrolovat, zda neexistuje průsečík horizontálních nebo vertikálních skoků ke stěně v daném směru a diagonálních os, vycházejících z cíle a opačně. Když skok ke stěně protne danou osu, musí se buňka přidat do množiny otevřených buněk.

Rychlost hledání cesty bez předzpracovávání mapy je tedy výrazně rychlejší než u Jump Point Search, ale vyplatí se až při hledání většího počtu cest na stejné mapě, protože se předzpracovaná data při každé změně mapy musí znovu předzpracovat. Každé předzpracování znamená několikrát projít mapu. Algoritmu Jump Point Search stačí mapu projít jen jednou.

Jump Point Search Plus má navíc mírně větší paměťovou náročnost. U každé buňky si musí uložit délku skoků nebo vzdáleností ke stěnám v každém směru. To dělá na každou buňku 8 hodnot informace navíc.

Tento algoritmus zrychluje hledání cesty asi 70 až 350 krát. [7, 8, 9, 10]



Obrázek 3.13: Příklad hledání cesty pomocí Jump Point Search Plus

3.5 Ohraničení cílů

Ohraničení cílů (Goal Bounding) je algoritmus pro prořezávání možných cest při hledání. Může se používat na obecné grafy, tak i na grafy reprezentující mřížkovou mapu. Algoritmus sám neumí najít nejkratší cestu, jen díky prořezávání snižuje počet vrcholů, které je nutné prohledat. A proto se musí kombinovat se jinými algoritmy.

Algoritmus pro každou hranu vycházející z vrcholu, zjistí obdélníkový prostor, ve kterém se nachází všechny optimálně dosažitelné vrcholy. Tvar tohoto prostoru může být jakýkoliv, ale jestli vrchol spadá do obdélníkového prostoru se testuje jednodušeji oproti složitěji tvarovaným oblastem.

Obdélníkový prostor se vypočítá tak, že se pro každý vrchol grafu spočítá Dijkstrův algoritmus. Tím na každé hraně tohoto vrcholu vznikne podgraf, který obsahuje všechny optimálně dosažitelné vrcholy přes tuto hranu. Poté se u každé hrany spočítá, do jaké obdélníkové oblasti se vrcholy tohoto podgrafu vejdou. Časová složitost tohoto algoritmu je tedy velmi velká $O(n^2)$. Navíc tento algoritmus zvyšuje paměťovou náročnost výpočtu. Pro každý směr si potřebuje uložit čtyři hodnoty vymezující vypočítaný obdélníkový prostor, ve kterém se nacházejí všechna řešení nejkratších cest pro tento vrchol jdoucí tímto směrem. Tedy pro mapy reprezentované osmisměrnou mřížkou, kde z každé buňky vede osm cest, to znamená 32 hodnot pro každou buňku.

Použití takto vypočítaných hodnot je velice jednoduché. Pokaždé když jiný algoritmus, použitý k hledání cesty, chce expandovat z jednoho vrcholu do dalšího, stačí zkontrolovat, jestli je cílový vrchol obsažený v obdélníkovém prostoru uloženém u této hrany. Pokud není, není třeba expandovat přes tuto hranu do dalšího vrcholu, protože přes něj nevede nejkratší cesta.

Tento algoritmus se tedy vyplatí používat spíše pro statické mapy, protože výpočet optimálně dosažitelných vrcholů stačící spočítat pouze jednou. Výpočet těchto hodnot, který je potřeba udělat při každé změně mapy, je kvůli kvadratické časové složitosti tohoto algoritmu velmi pomalý. Oproti tomu je hledání jedné nejkratší cesty velice rychlé. Vyplatí se tedy až při hledání velkého množství nejkratších cest na jedné mapě. Nebo pokud je mapa známa v době, kdy není ještě potřeba hledat nejkratší cesty, tak může tento algoritmus tuto mapu předzpracovat ještě před začátkem hledání cest.

Hledání pomocí A* v kombinaci tímto algoritmem je asi 20 až 60 krát rychlejší než samotný A*. Při kombinaci s Jump Point Search Plus pak asi 1400 až 5000 krát rychlejší než A*.[10]

3.6 Nafukování překážek

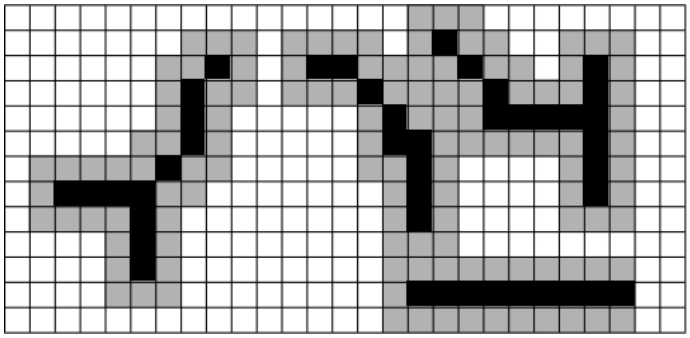
Nafukování překážek je algoritmus, kdy se překážky na mřížkové mapě rozšíří i do buněk, kde je volný prostor. Většina algoritmů počítá s pohybem po mapě

z buňky do sousední buňky. Velikost buněk je většinou menší než velikost robota a v reálném prostředí zabírá i více buněk. Při nafouknutí překážek tak robot získá bezpečný odstup od překážek.

Další výhoda této techniky je, že když jsou mapy vytvářeny laserovým skenerem, tak se v překážkách mohou vyskytnout malé mezery, které vznikají nepřesností skeneru nebo nevhodností skenovaného prostoru, který dobře neodráží laserové paprsky, jako je sklo. Tyto mezery se nafouknutím překážek mohou částečně eliminovat.

Nafouknutím překážek se navíc zmenší počet volných buněk mapy a tím se zrychlí hledání cesty. Také se tím vyhladí některé kostrbaté a nerovné hrany překážek, které mohou vzniknou při skenovaní a to je pro některé algoritmy jako je Jump Point Search nebo Jump Point Search Plus výhodné.

Nevýhodou této techniky je zkreslení délky trasy a plánovací algoritmus pak nemusí najít optimální trasu. Navíc je nutné zvolit vhodnou velikost nafukování, aby se neuzavřely některé užší průchody, jako jsou dveře v budovách a podobně. Tento algoritmus navíc potřebuje před začátkem hledání cesty samotným algoritmem projít celou mapu, aby mohl nafouknout překážky.



Obrázek 3.14: Nafukování překážek o jednu buňku. Původní překážky jsou černé, šedé jsou nové překážky po nafouknutí

Kapitola 4

Výběr algoritmu

Spolu s vedoucím práce jsme při výběru algoritmu brali v potaz požadavky na další použití jeho implementace. Algoritmus poběží jako uzel v prostředí ROS (Robot Operating System) nebo v simulovaném prostředí používaném na výzkumném pracovišti vedoucího práce.

Požadavky pro prostředí ROS jsou, že algoritmus poběží jako součást uzlu v ROS. Uzel v ROS získá vstupní data od dalšího uzlu s názvem *Gmapping*, který sbírá data z laserového skeneru robota. Laserová data zpracovává a vytváří virtuální mapu prostředí. Výslednou mapu odesílá ve formě mřížky přibližně každých pár vteřin.

Náš uzel na získané mapě nalezne buňky ležící na hraně mezi neznámým a volným prostorem. Z nich se vybere náhodně určité množství bodů (řádově stovky) a pro každou dvojici se spočítá délka nejkratší cesty. Tyto cesty se uloží do matice vzdáleností, která se odešle pro zpracování dalším uzlem v ROS.

Protože počet bodů, mezi kterými je potřeba hledat cestu, bude v řádů stovek, výsledný počet nejkratších cest, které je potřeba spočítat, vychází na desítky tisíc za každých pár vteřinu. Na nalezení jedné cesty tedy vychází maximální čas na desetiny milisekundy.

Po zvážení těchto kritérií, nepřichází v úvahu Dijkstrův algoritmus nebo A*. Ze zbylých vybraných algoritmů jsme nakonec vybrali algoritmus Jump Point Search Plus, protože je rychlejší než Jump Point Search na hledání jedné cesty a čas nutný k předzpracování mapy se v tak vysokém počtu hledání cest rozloží.

Algoritmus Subgoal Graphs jsme nevybrali, protože má delší čas pro předzpracování mapy, i když časy pro nalezení jednotlivých cest jsou lepší, ale i to nepřeváží delší čas pro předzpracování.

Navíc jsme nalezli open source implementaci s licencí BSD, která je optimalizovaná na rychlost předzpracování mapy, i následné hledání nejkratší cesty.

Tato vybraná implementace se ještě rozšíří o nafukování překážek, aby se ještě zrychlil výpočet.

Kapitola 5

Implementace Jump Point Search Plus

K implementaci Jump Point Search Plus jsem požil open source implementace, kterou publikoval Steve Rabin v roce 2015. [10] Tato implementace je optimalizovaná pro soutěž plánovacích algoritmů Grid-Based Path Planning Competition. [11]

Každý účastník v této soutěži musí implementovat jejich soutěžní API, které se skládá z několika funkcí. Nás budou zajímat tři základní funkce, které bude potřeba upravit: funkce pro předzpracování mapy, přípravy předzpracovaných dat k hledání cesty a funkce vracející nejkratší cestu mezi dvěma zadanými body.

V soutěži se předzpracovaná data ukládají do souboru s názvem předaným jako jeden parametr předzpracovávající funkce. Tento soubor si později načte funkce na přípravu předzpracovaných dat k hledání cesty. To je pro naše použití zcela nevhodné řešení, protože zápis a čtení z disku by zpomalovalo celý proces hledání cest.

Kód jsem tedy upravil tak, že zpracovaná data zůstávají v paměti počítače a funkce pro předzpracování dat vrací pouze referenci na ně. Z přípravné funkce se do funkce pro předzpracování dat ještě přesunul kód, který dopočítává směry, ve kterých je daná buňka blokovaná. Tento kód byl ve funkci pro přípravu dat k hledání cesty a ne ve funkci pro předpracovávání mapy nejspíše kvůli úspoře místa souboru, protože se dá velice lehce z ostatních předzpracovaných dat dopočítat.

Implementace obsahovala i algoritmus ohraničení cílů, který je pro naše použití moc pomalý a tak ho bylo potřeba odstranit.

Odstranit se musel na třech místech. Výpočet obdélníkových oblastí ohraničující cíle při předzpracování mapy. Dále místo vyhrazené pro takto vypočítané ohraničující oblasti v datové struktuře uchovávající předzpracovaná data. A samotné použití těchto dat v algoritmu hledající nejkratší cesty.

Tato implementace má navíc dvě optimalizace pro zrychlení prohledávání pomocí Jump Point Search Plus.

První z nich je rychlý zásobník (fast stack). Tento zásobník se kombinuje s množinou otevřených buněk a může se použít pouze, když se pracuje s ocile

heuristikou na mřížkových mapách. Protože když vyjde hodnota buňky menší než hodnota buňky, ze které jsme expandovali, pro všechny ostatní následující buňky, které se budou expandovat z této buňky, nemůže vyjít hodnota lepší, dokud není tato buňka zařazena do množiny uzavřených buněk.

Proto se nemusí tato buňka přidávat do množiny otevřených buněk, která je implementovaná jako prioritní fronta, která je výrazně pomalejší než zásobník. Dokud jsou nějaké buňky v tomto zásobníku, tak se pro další chod algoritmu berou z něj. Až když je zásobník prázdný, použijí se buňky z množiny otevřených buněk.

Druhou z optimalizací je vyhledávající tabulka (lookup table) funkcí pro expandování zpracovávaných buněk. Pro každou buňku je v předzpracovaných datech uložená informace, ve kterých směrech sousedí tato buňka přímo se stěnou v osmi bitové hodnotě (pro každý směr jeden bit).

Směr prozkoumávaní se dá uložit do tří bitů. Proto jde kombinací těchto dvou hodnot spočítat adresu funkce ve vyhledávací tabulce potřebné k prozkoumávaní sousedních volných buněk v daném směru. A u každé buňky se tedy nemusí zvlášť vyhodnocovat, jaké její sousedy prozkoumat.

Počet funkcí v této tabulce je $2^{8+3} = 2048$, jedna funkce pro každou kombinaci možných překážek a směrů. Zrychlení výpočtu pomocí této tabulky je v jednotkách až desítkách procent.

K takto upravené implementaci jsem doprogramoval nafukování překážek pro ještě rychlejší výpočet, s množností volby o kolik buněk se na vstupní mapě budou nafukovat překážky.

5.1 ROS - Robot Operating System

ROS je robotický operační sytém (Robot Operating System). Jo to sada softwarových balíčků pro práci s hardwarem tak i softwarem pro řízení a zpracování dat. Základem tohoto sytému jsou tzv. uzly, které mohou přijímat a odesílat zprávy.

V této práci nás zajímá uzel *Gmapping*, který přijímá data z laserového skeneru a vytváří dvojrozměrné pole, které reprezentuje mapu prostředí. V každé buňce je hodnota od 0 do 100 vyjadřující míru pravděpodobnosti, že je v dané buňce mřížky překážka. Pokud je stav buňky neznámý, tak je hodnota -1. Tuto mapu s dalšími údaji, jako je výška a šířka této mapy, odesílá dalším uzlům.

Mnou vytvořený uzel tyto data přijme. Zjistí šířku a výšku mapy, kterou si uloží pro další použití. V prvním kroku se vytvoří ze vstupních dat mřížková mapa. Jako volná místa se označí ty buňky, u kterých je pravděpodobnost překážky menší než 10% nebo pokud jsou neznámé. Na takto vytvořené mapě provede nafukování překážek o zvolenou hodnotu.

Poté se projde nafouknutá mapa a zjistí se, které volné buňky sousedí s neznámým prostředím v původních datech. Z takto nalezených buněk se

náhodně vybere zvolený počet buněk, mezi kterými se budou hledat nejkratší cesty.

Pak se nafouklá mapa předá algoritmu Jump Point Search, který provede předzpracování mapy. Následně se pro všechny dvojice náhodně vybraných hraničních buněk spočítají nejkratší cesty, které se uloží jako matice vzdáleností.

V posledním kroku se takto vytvořená matice odešle dalším uzlům a uzel čeká na další vstupní data.

5.2 Simulované prostředí

Vybraný algoritmus bylo potřeba implementovat do simulačního prostředí, které se používá na výzkumném pracovišti vedoucího práce. Toto prostředí využívá hledání nejkratších cest k nalezení optimální trasy k prozkoumání neznámé mapy.

Program simulující prohledávání neznámé mapy vybere na hraně neznámého prostředí a volných míst vhodné body, které má robot navštívit. K těmto bodům přidá svojí polohu a spočítá nejkratší cesty mezi nimi. Poté pomocí takto nalezených nejkratších cest řeší problém obchodního cestujícího (Travelling Salesman Problem – TSP), tedy hledá nejkratší možnou cestu přes všechny požadované buňky.

V tomto prostředí tedy bylo nutno upravit funkci, která hledá nejkratší trasy mezi zadanými uzly. V původní implementaci byl použit Dijkstrův algoritmus, který je v tomto případě pomalejší na mřížkových mapách než Jump Point Search Plus.

Mapu ve formě mřížky používala již předchozí implementace. Takže ji nebylo potřeba dále upravovat. Tato mřížka se tedy přímo použije pro předzpracování v Jump Pont Search Plus algoritmu. A na výsledných datech se hledají nejkratší cesty mezi požadovanými buňkami. Výsledky se uloží do matice vzdáleností a předají algoritmu pro řešení problému obchodního cestujícího.

Kapitola 6

Testování

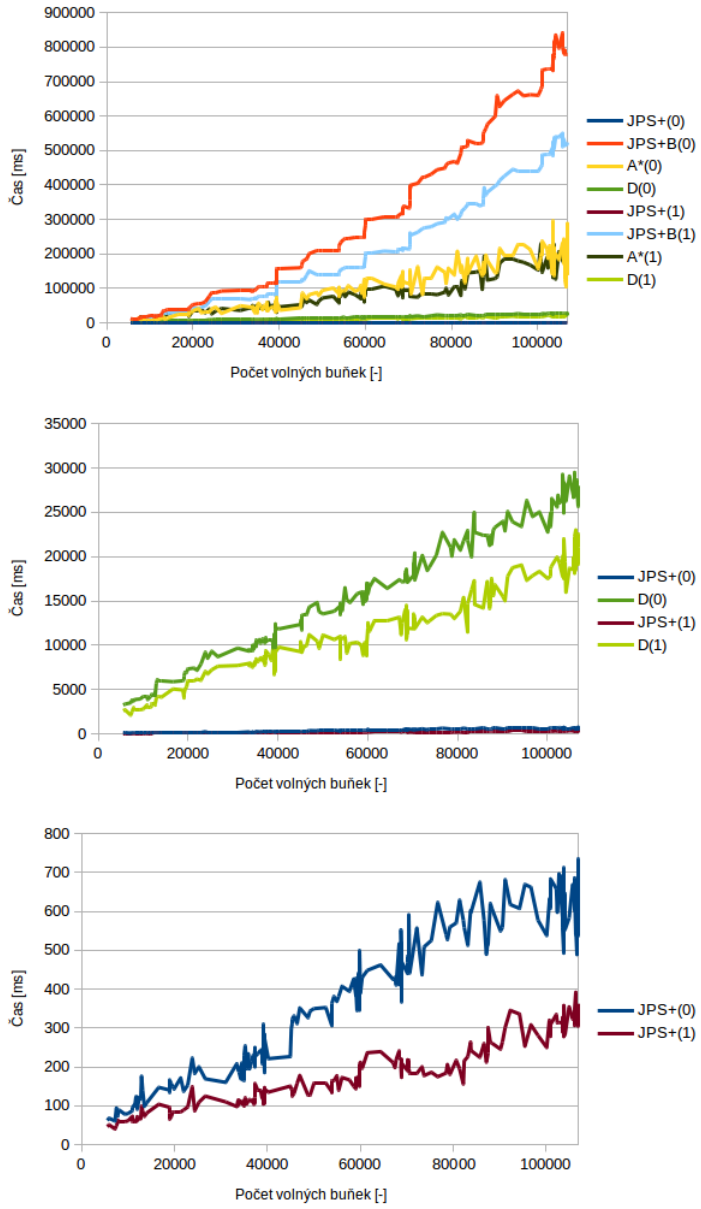
Implementaci Jump Point Search Plus budu testovat v porovnání s těmito algoritmy: Dajkstrův algoritmus, A*, Jump Point Search Plus a Jump Point Search Plus s ohraničením cílů, všechno v kombinaci s nafukováním překážek o žádnou buňku a o jednu buňku. Větší nafukování by již pro testované mapy zamezovalo průchodům v některých částech mapy.

Jako vstupní data použiji data z reálných robotů. Záznamy obsahují laserové skeny dvou pater z různých budov. Mapy se vytvářely pomocí uzlu *Gmapping*. Pro měření jsem upravil mnou implementovaný výpočetní uzel tak, aby na mapě postupně spustil všechny algoritmy pro oba stupně nafouknutí překážek a změřené časy uložil.

Čas je měřen od začátku předzpracovávání mapy až do spočítání všech nejkratších cest mezi vybranými buňkami. Počet buněk, pro které se počítají nejkratší cesty, jsem stanovil na 100. Je tedy potřeba, při každém obnovení mapy, spočítat 4950 nejkratších cest.

Při měření se objevil jeden nový problém. ROS po jedné hodině odstraňuje stará data z fronty zpráv pro zpracování. Protože výpočet pro všechny algoritmy trvá několik hodin, tak se měření po jedné hodině zastaví. Musel jsem tedy doprogramovat jeden uzel, který se chová jako buffer, všechna data uloží a na požádání je posílá dál.

Obrázek 6.1: Kompletní sken části jednoho patra v budově Blox v Dejvicích



Obrázek 6.2: Grafy času běhů algoritmů v závislosti na počtu volných buněk před nafouknutím mapy

Jako datové podklady pro první měření jsou použity laserové skeny části jednoho patra v budově Blox v Dejvicích.

Příprava mapy, nafukování překážek a výběr náhodných bodů trvá maximálně $5,2$ *ms* a není v měřených časech zahrnuto. Uzel *Gmapping* posílá aktualizované mapy přibližně každých $5,5$ *s*.

Na grafech je vidět, jak dlouho trval výpočet v závislosti na počtu buněk před nafouknutím mapy. Z dat vyplývá, že algoritmus Jump Point Search s ohraničením cílů a A* jsou pro naše požadavky zcela nepoužitelný. Čas $5,5$ *s* překročily téměř hned na začátku průzkumu neznámého prostoru a to bez

ohledu na hodnotu nafukování překážek. Dijkstrův algoritmus přestal stíhat po 99 s prozkoumávání prostoru bez nafukování a s nafukováním překážek o jednu buňku až po 154 s prozkoumávání. Nesmíme zapomenout, že hledání cest je jen přípravou dat pro rozhodování robota při průzkumu neznámého prostoru, proto je i Dijkstrův algoritmus nepoužitelný, protože robot by se po pár desítkách sekund začal sekat.

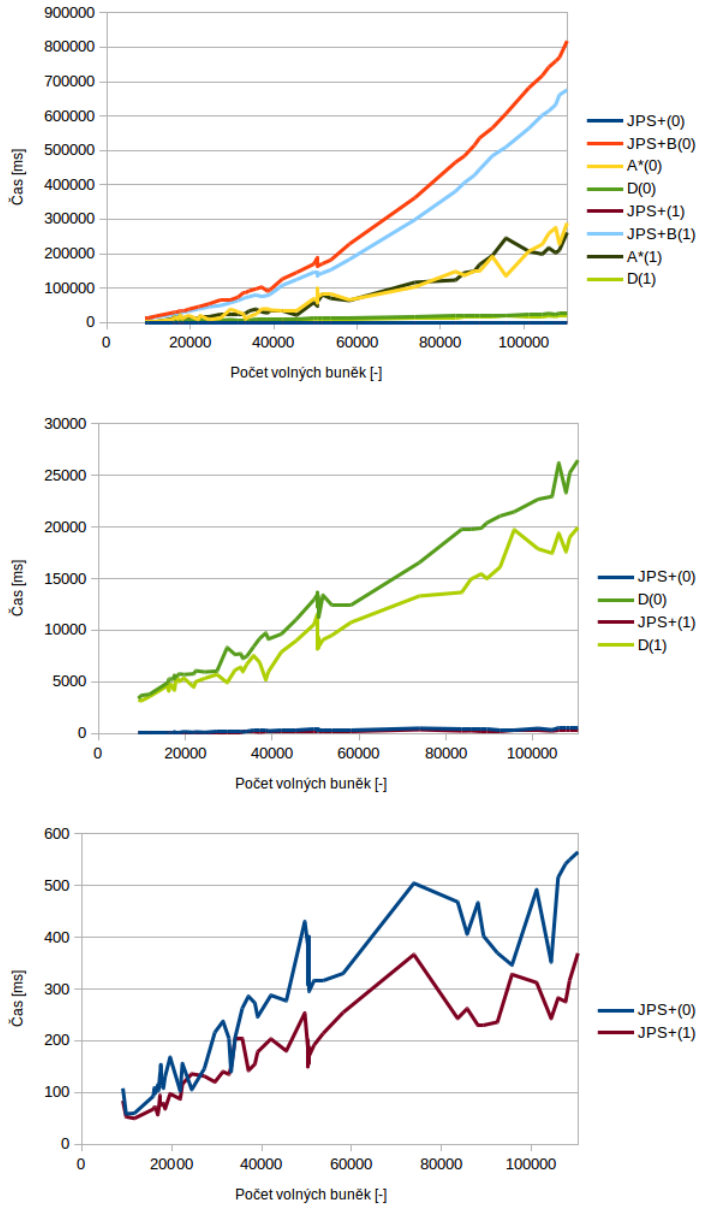
Nejlépe splňuje naše požadavky algoritmus Jump Point Search Plus bez nafukování překážek mu hledání cest trvalo maximálně $884,6$ ms a s nafukováním o jednu buňku jen 416.7 ms.

Po výpočtu koeficientu lineární regrese vychází koeficient $0,00588$ pro případ bez nafukování a $0,00274$ pro případ s nafukováním o jednu buňku. Při zvážení této aproximace bude délka výpočtu trvat $5,5$ s při 935436 volných uzlech v prvním případě a 2004307 v druhém případě. Hustota volného prostoru u této mapy je $12,44\%$. Po přepočtu na rozměry podobných map to odpovídá $2742x2742$ nebo $4013x4013$.

V případě, že bychom potřebovali vše spočítat do jedné sekundy, aby zbyl čas na rozhodování, vychází hodnoty 170079 volných buněk a velikost mapy $1169x1169$ pro případ bez nafukování překážek. V případě s nafukováním o jednu buňku je to 364419 volných buněk a mapa o rozměrech $1711x1711$.

Pro druhé měření jsou použity laserové skeny části jednoho patra v budově CIIRC v Dejvicích.

Obrázek 6.3: Kompletní sken části jednoho patra v budovy CIIRC v Dejvicích



Obrázek 6.4: Grafy času běhů algoritmů v závislosti na počtu volných buněk před nafouknutím mapy

Z grafů je v této situaci vidět, že grafy jsou více kostrbaté. To je způsobeno tím, že laserový sken není tak dobrý. Ve stěnách jsou občas mezery a také není tak dlouhý.

Výsledky měření jsou ale stejné, jako v předchozím případě. Všechny testované algoritmy nejsou dost rychlé, až na Jump Point Search Plus. Dijkstrův algoritmus v tomto případě přestane stíhat již po $49,5\ s$ v případě bez nafukování překážek a po $93,5\ s$ v případě, že jsou překážky nafukovány o jednu buňku.

Maximální doba nutná pro zpracování mapy byla $564,74\ ms$ v prvním

případě a $369,25\ ms$ v případě druhém.

Koeficienty lineární regrese vyšli $0,00433$ a $0,00249$. Hustota volného prostoru této mapy je $18,84\%$. Odhad maximálního počtu volných buněk, který by se dal prozkoumat na podobné mapě za dobu $5,5\ s$, poté vychází 1269893 při rozměrech mapy $2596x2596$ pro případ bez nafukování překážek a 2211949 volných buněk při rozměrech mapy $3426x3426$.

V situaci, kdy potřebujeme spočítat nejkratší cesty do jedné sekundy, vychází odhad na 230889 volných buněk a velikost mapy $1107x1107$ pro první případ a 402172 volných buněk a velikost mapy $1461x1461$ pro druhý případ.

Kompletní záznamy se změřenými časy jsou přiloženy v tabulkách A.1 a A.2.

Pro měření byl použit počítač LENOVO ideapd Y700, který má procesor Intel Core i5-6300HQ CPU @ 2.30GHz a operační paměť 8GiB @ 2133MHz.

Kapitola 7

Závěr

Při prohledávání neznámého prostoru je potřeba co nejrychleji hledat nejkratší cesty, které se potom používají k plánování prohledávání. Plánování probíhá velice často a proto není moc času na samotné hledání cest. V případě, že si robot uchovává mapu prostředí ve formě mřížky, se vyplatí používat specializované algoritmy, které dokáží využít jejich vlastností.

Takový algoritmus je například Jump Point Search, který je spíše použitelný pro hledání několika málo cest. V případě hledání většího množství cest se vyplatí využívat jeho vylepšenou verzi Jump Point Search Plus, která si mřížkovou mapu předzpracuje a tyto informace poté využívá k zrychlení prohledávání mapy. Oba tyto algoritmy využívají symetričnost mřížky. Další možností je využívat místa na mřížce, přes které musejí jít nejkratší cesty, jako jsou rohy překážek. Tohoto principu využívají algoritmy grafů podcílů.

Použití obecnějších algoritmů, jako je A* nebo Dijkstrův algoritmus, se nevyplatí, pokud není prostor mřížky velice malý nebo pokud není potřeba spočítat nejkratší vzdálenosti jen mezi několika vrcholy. Pokud se počítá vzdálenost jen mezi 10 místo 100 vrcholy, je poměr délky času výpočtu pro algoritmus A* 110 násobný. Výhodou těchto algoritmů je pak jejich jednodušší implementace.

Z těchto vyjmenovaných algoritmů jsme vybrali Jump Point Search, protože je na mřížce pro naše využití dostatečně rychlý. Nevybrali jsme algoritmus grafu podcílů, i když hledá jednotlivé cesty rychleji. Čas nutný na předzpracování mapy je pro naše použití moc velký.

Nakonec jsem našel open source implementaci Jump Point Search Plus s ohraničením cílů a licencí BSD. Tato implementace je velice optimalizovaná na rychlost výpočtu nejkratších cest. Z algoritmu byla odstraněna část ohraničování cílů, protože byla pro naše účely moc pomalá. Tento upravený algoritmus byl využit v ROS a simulovaném prostředí.

Další možné rozšíření této práce by se mohlo věnovat paralelizování některých výpočtů, odstranění nějakých opakovaně prováděných alokací a dealokací paměti nebo znovu implementovat vhodně upravený algoritmus ohraničování cílů pro požadavky našeho použití.

Literatura

[1] *Nejkratší cesty*. 30. ledna 2015 [online] http://mj.ucw.cz/vyuka/ga/13-dijkstra.pdf [citováno 8.1.2018]

[2] Amit Patel: *Heuristics*. 28. listopadu 2017 [online] http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html [citováno 8.1.2018]

[3] Tansel Uras, Sven Koenig a Carlos Hernández: *Subgoal Graphs for Optimal Pathfinding in Eight-Neighbor Grids*. Department of Computer Science, University of Southern California, Los Angeles, USA a Depto. de Ingeniería, Informática Univ. Católica de la Ssma. Concepción, Concepción, Chile 2014 [online] https://www.aaai.org/ocs/index.php/ICAPS/ICAPS13/paper/view/6058/6182 [citováno 8.1.2018]

[4] Andreas Thüring: *Subgoal Graphs for Fast Optimal Pathfinding*. Seminar: Search & Optimization Universität Basel, 22. října 2015 [online] http://ai.cs.unibas.ch/_files/teaching/hs15/search-opt-seminar/slides/slides_Andreas_Thuering.pdf [citováno 8.1.2018]

[5] Tansel Uras a Sven Koenig: *Identifying Hierarchies for Fast Optimal Search*. Department of Computer Science, University of Southern California, Los Angeles, USA, 2014 [online] http://idm-lab.org/bib/abstracts/papers/aaai14a.pdf [citováno 8.1.2018]

[6] Nathan Witmer: *Jump Point Search Explained*. 5. května 2013 [online] https://zerowidth.com/2013/05/05/jump-point-search-explained.html [citováno 8.1.2018]

[7] Daniel Harabor a Alban Grastien: *Improving Jump Point Search*. NICTA and The Australian National University, 2014 [online] http://users.cecs.anu.edu.au/~dharabor/data/papers/harabor-grastien-icaps14.pdf [citováno 8.1.2018]

[8] Daniel Harabor a Alban Grastien: *The JPS Pathfinding Syste.* NICTA and The Australian National University, 2012 [online] `http://users.cecs.anu.edu.au/~dharabor/data/papers/harabor-grastien-socs12.pdf` [citováno 8.1.2018]

[9] Daniel Harabor a Alban Grastien: *Online Graph Pruning for Pathfinding on Grid Maps.* NICTA and The Australian National University, 2011 [online] `http://grastien.net/ban/articles/hg-aaai11.pdf` [citováno 8.1.2018]

[10] Steve Rabin: *JPS+ with Goal Bounding.* commint 21e362a1edfe7e6e0a70358fd41117fc8c27a0ca, 22. březen 2015 [online] `https://github.com/SteveRabin/JPSPlusWithGoalBounding` [citováno 8.1.2018]

[11] Nathan Sturtevant: *GPPC: Grid-Based Path Planning Competition.* SUniversity of Denver, 2014 [online] `http://movingai.com/GPPC/cfp.html` [citováno 8.1.2018]

Příloha A

Tabulky

Tabulka A.1: Tabulka časy pro budovu Blox

	JPS+(0)	JPS+B(0)	A*(0)	D(0)	JPS+(1)	JPS+B(1)	A*(1)	D(1)
5511	62,289	12378,4	2061,53	3256,8	46,117	9093,04	7281,25	2600,48
5932	67,912	11363,4	3874,77	3341,61	50,683	8799,52	7413,62	2702,03
7178	60,456	11637	5370,61	3495,26	40,049	9057,64	6747,6	2101,57
7489	93,778	13499,2	14511,6	3803,41	48,312	10656,5	10785,7	2486,01
7851	73,957	14800,8	8956,15	3766,3	58,723	12237,1	15700,6	2946,13
7922	79,11	17640,2	6678,28	3703,11	62,731	13507,9	16272,5	2670,14
8112	88,469	17454,3	10178,7	3860,49	58,163	14216,4	13880,2	2686,74
9278	78,027	18431,1	6169,03	3920,84	58,667	14481	17612,9	2766,12
10002	79,81	19424,4	7220,46	4156,04	61,201	15204,8	18596,9	2785,02
10819	87,142	19347,5	7488,42	4176,64	72,658	15143	15412,9	3238,97
10966	94,825	18971,1	8463,42	3855,28	59,888	15456,4	12678,9	3000,48
11505	104,83	18944,3	12510,6	4248,24	58,843	15647,2	15727,9	3037,37
11614	122,994	20119,1	13074,6	4154,74	62,923	15835	18590,3	3010,26
11728	98,035	20636,3	8869,25	4343,46	59,786	16268,3	14613,9	3108,51
11841	122,262	20961	12309,6	4430,04	71,567	16525,3	18681,5	3268
11879	90,834	21087,3	7250,26	4464,82	60,907	16628,6	14906,4	3320,97
12396	120,511	21460,3	14135,3	4331,85	68,188	17206	17161,7	3454,72
12626	117,663	22243,4	14720,3	4700,31	71,231	17963	19371	3237,44
12726	117,969	23498,3	11980,3	4323,51	66,448	18620,2	17710,2	3388,48
12896	175,76	29944,7	17763,5	5696,02	98,599	24374,3	24802,3	4002,9
13125	138,783	35919,1	16907,1	6102,76	85,72	29039,4	20569,4	4158,37
13520	98,189	36264	9183,09	5958,11	72,609	28812,9	22133,2	4219,45
14012	107,263	36739,7	13570,9	5931,1	81,166	28735,9	25427,5	4118,03
16612	147,226	38112,6	24038	5874,42	103,617	29952,1	37554,8	5052,46
18818	140,084	37986,1	25667,2	5981,31	95,765	30675,8	34006,9	4934,51
19020	134,038	38527,8	20857,8	6172,3	80,727	31003,3	33185,5	4050,77
19041	166,18	42402,8	32573,4	6505,76	65,574	33577,8	18453,3	4461,15
19261	156,951	44217,4	31446	6907,93	71,004	34988,6	21800,4	5009,84
19628	148,739	48558,2	29260,6	6859,82	82,319	36867,3	28755,7	5270,82
19901	142,984	52331,9	34263,5	7316,21	83,681	41284,5	33289,5	5976,44
21312	171,402	54268,6	41244,7	7415,75	83,135	42273,1	35447,8	5931,23
21952	137,089	56896,1	28716,5	7157,86	89,053	45194,1	37687,6	6165,85
22831	152,896	60762	32902,5	7864,68	96,456	48295,1	28981,1	6028,45
23812	222,663	77557	31899,8	9205,59	148,642	61928	39855,2	7035,19
24331	182,951	86344,5	40339,5	8522,69	86,858	69815,5	25652,2	6747,98

	JPS+(0)	JPS+B(0)	A*(0)	D(0)	JPS+(1)	JPS+B(1)	A*(1)	D(1)
25244	199,978	87198,8	45446,4	9313,93	108,963	69494,6	33682,5	7222,96
26605	169,07	92108,9	27068,4	8698,24	124,598	70128,6	41417,1	7619,33
31008	160,119	94864,2	49245,2	9664,57	109,722	68819	35892,9	7699,95
33391	207,809	92917,3	44005,5	9327,93	97,8	67259,9	43736,8	7938,4
33627	192,238	91850,9	45596	9566,1	114,202	68092,2	42399,9	7722,48
33782	200,145	92145,2	47012,7	9464,68	100,365	68799	42650,1	7653,88
33889	186,022	93198,4	41888,3	9311,8	113,667	69045,5	49910,6	7961,15
34249	169,127	94533,8	28615,4	9356,91	110,751	69356,3	36635,6	7676,39
34428	209,83	94775,7	42747,3	9976,6	104,582	71879,2	41783,6	7512,37
34518	186,142	98265,6	34150,8	9565,92	107,937	69417,6	49189,2	7744,02
34665	186,414	97263,1	38327,8	9619,47	109,364	69655,3	41709,1	7893,23
34742	166,273	96939	31744,1	9664,9	110,724	70619,2	41755,7	8133,52
34878	164,888	102344	28523	10137,4	99,353	74717,1	32776,2	7789,91
35040	230,45	102349	51577	9844,93	109,84	74046,6	44784,7	8291,79
35206	253,934	104974	56545,9	10574,6	120,41	76082,6	45996,4	8443,72
35540	195,045	104183	51547,4	10342,2	104,456	78383,5	40311,5	8152,02
35847	233,528	105286	45711,7	10619,7	113,203	77037,6	40191,8	8415,89
36121	194,919	105931	37785,9	10226,5	115,982	78221,6	47024,1	8566,69
36577	212,863	105272	50758,8	10711,2	110,854	78379,6	42446,7	8226,31
36865	228,706	105775	48599,3	10453,8	115,605	77674,2	45243,9	8704,22
37076	212,378	106497	34869,2	10240,7	103,551	77047,2	38757,2	7747,68
37195	243,803	105271	59137,1	10499,4	106,728	76809,7	36948,5	8176,12
37181	199,842	108045	44374,4	10653,5	121,085	77611	50363,2	8169,04
37231	231,574	111071	37092,9	10848,1	142,395	81187,9	49339,2	8261,93
37351	249,701	115754	35277,5	10621,6	136,953	82484,8	48322,7	8612,8
37344	227	116140	39275,3	10419,1	157,053	82839,1	59318,6	9366,47
37904	233,326	113668	42261	10631,9	140,066	82901,3	44345,8	8957,71
38512	242,523	113900	45990,2	10563,6	138,902	83102,9	50506,9	8285,34
38942	244,383	116708	41272,7	10337,7	119,247	83284	38069,7	9624,89
38980	208,303	116606	31476,6	9700,42	135,704	83283,2	39805,5	8884,17
39022	218,992	114709	39523,3	10076,4	127,615	82968,1	42616,9	8441,66
39076	309,828	114150	62308,8	10550,1	125,452	82974,2	41255,8	8835,76
39106	267,009	116882	56213,3	11142,6	126,313	83220,2	41264,9	8665,83
39116	252,317	116918	47915,5	10322,7	118,225	83440,3	37962,3	8090,95
39129	214,391	114258	38467,1	10621,3	120,668	79704,5	51341,5	7687,32
39168	219,471	114674	44998,7	11051	119,467	79934,5	48788,2	7526,08
39160	281,891	117373	57132,7	11222,4	104,338	79991,5	42808,2	6687,22
39210	239,063	117358	39855,9	10716,5	131,695	79977,2	48021,4	7629,74
39218	277,617	114754	59958	10928	119,407	81427,8	51811,5	7083,71
39310	185,252	114451	33412,2	9745,42	110,941	79793,3	42297,2	7054,66
39324	204,072	114927	38727,2	10579,4	123,542	83457,7	40948,4	8226
39330	218,624	114902	47536,7	11109,2	106,199	83475,7	30179,5	8300,1
39332	252,919	117451	54374,8	10280,9	113,832	79960,8	47786	7645,25
39310	202,434	117450	35586	10458	122,402	79713,9	51544,8	7596,95
39313	240,701	115092	41696,4	10746,7	124,105	79823,3	51638,7	7558,44
39314	268,392	115651	55990,4	10709,2	112,653	80226,9	43509,1	7241,15
39308	226,401	115800	37666,6	10737,2	114,381	80341,3	49228,1	7710,83
39312	231,565	118840	44185,3	10849,8	108,449	82757,3	41604,9	7235,74
39336	283,452	156337	53747,4	11777,4	134,585	112594	54912,5	8338
39451	279,014	157259	73249,2	12391	146,554	117551	51480,6	9604,47
39500	272,022	157650	60642,5	11947,1	142,766	117967	49017,6	9211,57
40175	221,028	158214	36401,9	11835,1	134,712	118292	46270,7	9776,39
44904	226,538	159552	43346,2	12339	150,373	119335	51219,4	9269,44
45122	299,218	163780	74386,5	12760,4	142,791	121485	51699,5	10388,6
45299	311,611	166982	82717,4	11712,1	142,821	124838	47133,3	9679,16

	JPS+(0)	JPS+B(0)	A*(0)	D(0)	JPS+(1)	JPS+B(1)	A*(1)	D(1)
45401	323,394	177566	86412,2	13354,1	124,647	122830	53516,8	9195,98
45710	331,117	180651	80705,3	13401,6	131,84	124172	71269,8	9787,1
46364	311,248	184680	69467,6	13465,9	157,285	137862	58703,4	9970,56
46948	351,681	200505	81397,7	14294,1	177,847	150533	65821,2	11180,1
48719	326,611	209303	87568,4	14813,6	126,949	139367	53031,8	10453,1
49159	343,822	210206	78726	13784,5	127,383	138982	62635,2	9644,74
50016	350,008	209600	93820	13530,7	159,582	139297	71711,3	11141,4
52541	352,746	209902	100894	13837,9	157,012	139900	76580,2	10628,8
53887	306,174	210113	82535,6	14442,1	135,797	139490	57428,7	10996,2
53933	352,682	209852	84974,9	13782	140,585	139168	64336,7	10899,6
53936	357,217	209954	102854	13635,2	150,118	138933	64640,7	9273,01
53937	356,054	216603	79879,5	14412,2	153,206	139666	75265,7	9481,17
53969	325,771	216741	89694	14504,9	150,476	141747	65945,5	8659,15
53964	330,126	219178	75692,4	13450	133,831	143164	64094,8	8407,99
53966	363,395	223843	75259,1	14727,9	163,952	146389	82033,4	10713,8
54296	381,23	233448	93557,2	13908,2	163,363	154417	77319,9	10696,9
54922	368,599	241723	91804,7	16477,5	176,976	157319	92357,1	10964,9
55317	381,157	245021	93100,5	15114	139,993	159648	75823,1	9108,14
56031	407,479	245791	107341	14817,7	172,558	159639	89102,9	10192
57636	394,463	248501	91051,3	15806,7	166,47	160266	81390,2	10318,4
58558	426,093	246438	96885,9	16000,4	150,157	160517	73442,9	10006,7
59011	383,85	246552	108034	14623,5	143,504	160532	69812,4	8852,08
59051	427,718	249449	101869	15467,6	171,448	161623	82419,7	9798,48
59075	402,986	247840	89553,2	16085,2	154,169	161487	92508,6	9669,75
59278	345,183	249138	85443,8	15975,1	162,736	161511	70529,7	10266,7
59442	378,341	248083	96778,2	14956	174,88	161594	94235,8	9830,13
59651	440,307	247265	125727	15224,3	163,278	163012	70607,7	9349,23
59650	366,761	247693	80510,2	15470,4	160,86	162552	73640,7	8814,54
59699	345,973	246768	73951	15649,5	164,238	162880	81531,8	9658,16
59707	364,077	247155	104080	15621	157,285	162792	61523,7	9937,69
59794	406,994	249023	106383	15935,1	168,524	164295	72478,4	10042,5
59799	433,872	249176	111374	16100,8	181,871	166689	90582,4	9845,13
59796	408,016	253758	98927,2	16990,3	184,159	166136	74421,3	9675,18
59795	411,119	251755	93611,1	15526,4	150,072	165192	60916	8981,87
59846	381,79	258652	88649,2	15243,8	167,113	162778	79861,1	10094,5
59881	387,748	249759	99478,1	15666,8	178,603	167484	74803,6	10369,8
59881	355,321	261484	96109,5	15604,6	170,955	169028	92193,4	10089,5
59908	499,413	284140	119320	16958,8	206,75	192609	80243,1	11956,6
60005	390,308	298904	101528	15788,2	211,438	200746	99906,6	12515,1
60379	430,383	299304	128003	16568,7	196,35	202243	96126,7	11572,9
61496	448,497	300831	128353	17517,7	236,343	203654	95963,1	12734,6
64351	462,431	306419	112228	16414	239,141	207916	106018	12788,7
67032	425,224	307694	103642	17399,3	210,945	207346	97023,7	13151,5
67442	430,153	308569	96955,2	17248,2	198,191	207570	78290,8	11182,1
67687	410,648	310234	112915	17141,8	218,534	213220	92968,9	12510,4
68404	516,64	316326	138037	17213,2	241,075	214271	91130,5	13965,4
68448	411,831	317534	103190	18187,1	223,047	215366	83454,3	13963,8
68512	474,829	318311	122962	18485,3	219,089	215128	99656,1	13192,3
68565	492,588	318353	132631	18027,4	220,842	215336	89772,5	14490,8
68726	461,313	320273	125088	18574,7	217,425	216462	98766,1	13199,8
68734	367,193	316406	112660	17206,5	196,163	212866	80130,3	11111,7
68728	491,592	315745	138533	17972,9	172,971	212897	72844,7	10619,2
68720	552,585	330369	148553	18026,7	203,112	210090	85903,4	12294,3
68754	424,902	325448	105730	17592,6	219,262	211185	91554,6	13873,7
68741	518,67	333733	124529	17513,7	192,014	214710	95966,2	11376,4

	JPS+(0)	JPS+B(0)	A*(0)	D(0)	JPS+(1)	JPS+B(1)	A*(1)	D(1)
68932	470,903	338497	117680	17112,2	209,549	216121	93417,1	11910,7
70160	440,46	332581	113314	17887,4	190,191	213595	95021	11854,2
70242	467,731	337234	118778	19170,4	217,895	219233	96776,9	13293,7
70257	484,54	355904	111083	18180,6	206,201	225478	90640,8	13308,9
70322	479,959	396851	126929	18871,6	217,157	260375	76287,2	13518,9
70345	591,445	386897	153063	20400	195,201	253433	80836,7	13110,8
70601	441,749	399977	83564,5	17364,4	184,112	254079	75923,4	11868,7
72148	557,248	404719	162657	20121,2	183,698	262778	74201,6	13157,1
73259	437,301	423795	82046,8	18436,8	200,106	273268	87996,8	12537,1
73707	508,872	423678	130769	18792,4	177,4	275510	83587,7	12721,4
75272	525,678	431391	128170	20160,1	186,019	278653	82513,7	13351,3
76643	623,424	444305	161635	22740,9	174,468	287420	81392	13551
78444	540,932	449095	146042	21054,7	184,161	291796	88754	13491,5
78729	527,809	457912	114088	20060,3	205,977	304329	100101	13347,6
79264	559,864	464113	158219	21894,9	180,576	301264	80524,7	13003,1
80626	570,49	467989	138666	20728,4	216,728	315177	103923	13821,3
81343	629,257	463425	207322	21783,2	194,916	300424	105089	15381,9
82149	577,86	491503	146669	22946,2	155,825	313648	79217,6	11487,8
82338	559,016	511499	166064	21857	215,461	323362	121784	14674,4
83113	513,093	509021	115935	19978,7	226,139	333050	98247,3	16245,6
83712	601,879	514284	186812	24989,4	263,564	342839	139999	17275,8
83810	598,347	528977	143431	22749,6	241,823	344262	144653	14601,6
85710	675,595	520279	193588	22354,8	224,437	344960	147290	14200,8
86607	578,353	518969	152703	22373,5	259,753	339407	150497	17179
87114	489,867	522332	137053	21268,2	211,996	341566	96888,8	14138,3
87356	528,687	526256	141277	22138,9	235,497	343624	132865	14652,5
87378	524,62	549249	174185	22383,4	271,564	359293	154738	15696,3
87575	515,694	554544	144330	21416,2	300,96	391913	192478	17520
87960	620,595	564273	196052	22998,5	262,958	371238	170115	15608,3
88328	606,425	576421	180481	23306	260,719	379276	123901	16811,4
90138	549,804	599457	144520	23990,1	245,547	397516	128322	15475,1
90561	559,24	659166	146857	22912,7	263,973	400628	132986	15021,5
91161	681,773	627891	215790	25091,2	301,41	413397	173281	17779,4
92248	617,806	645182	195520	23931,8	345,502	425781	184314	18751,9
94236	607,846	663249	196047	23396,4	336,033	445583	185151	19042,8
95402	669,666	672928	225400	26336,3	253,283	440244	179523	17321,3
96700	662,29	658736	227352	24520,4	308,172	440840	175485	17847
98224	577,537	662062	212883	25043,6	280,75	440896	167744	18337,2
100090	538,172	660192	156981	22781,3	249,689	441437	152999	17520,3
100686	625,066	677588	191414	24911,3	319,909	451951	229298	17820,6
100743	632,34	680429	198772	23389	317,222	453925	178117	18605
100885	608,549	685041	197140	25133,1	278,453	456979	147031	18893,6
100930	683,636	733537	236781	26552,1	309,23	487050	147195	18903,7
102164	658,433	735731	213214	25597,6	334,805	488395	189404	19945,2
102377	597,985	735939	187057	26921,2	313,185	488000	159082	19826,2
102737	696,664	735200	223414	26068,2	312,962	488446	194404	19235,1
103379	584,913	735411	142064	26599,8	315,131	506854	147838	17787,8
103475	691,783	753821	220392	28209,5	312,505	503567	182367	19527,9
103484	647,09	766722	295812	29277,8	289,985	521133	178470	19193,9
103522	665,348	776972	214632	27215,5	302,232	505870	190004	20134,5
103506	677,287	757089	228164	26231,9	327,136	511792	187905	19284
103502	631,957	757048	205077	29113,6	292,488	503107	171050	18849
103522	586,498	748714	214308	26890,7	297,353	524613	168730	17548,3
103526	690,796	748065	230964	27773,4	303,804	486661	131010	20280,4
103524	557,491	736353	188674	25875,6	310,757	486965	174891	19421,4

	JPS+(0)	JPS+B(0)	A*(0)	D(0)	JPS+(1)	JPS+B(1)	A*(1)	D(1)
103646	493,004	732273	166261	24930,7	299,649	501808	182764	17976,2
103647	712,965	761968	211778	27086,3	323,314	507950	165788	19689,4
103689	497,678	769892	136716	28265,3	286,062	504843	170314	17580,5
103735	508,076	775523	158057	26434,9	304,536	513304	158308	19252,6
103736	560,838	785008	155273	28355,7	277,819	522913	138326	18914,8
103725	618,103	816165	218459	26288,9	358,014	537117	226634	21999,3
103931	645,892	790965	185398	28195,6	307,996	508036	208649	19427,2
104145	553,062	835123	200052	26266,1	282,403	539926	126758	15991,1
104919	583,542	797093	182840	29085,8	354,349	538826	206048	18691,7
105669	668,916	841604	232111	26869,8	323,237	550076	186274	18544,1
105890	661,306	800303	212000	26698,8	334,875	511133	178919	19891,9
105967	602,262	784649	181002	28075,8	345,63	518496	187669	18910,1
106029	611,226	788215	171862	27618,8	311,564	518253	182843	20977,8
106116	686,242	783148	242843	29500,5	306,586	533163	193021	18123,7
106194	655,156	785523	166020	26766,9	347,352	518471	225491	22253,6
106241	540,582	786381	138400	27992,5	391,865	519928	235580	23009,9
106283	629,055	790907	216184	27909,3	311,994	514932	191315	18495,3
106404	559,514	779768	124716	27611,6	360,433	515562	162220	22520,6
106476	489,138	780787	104412	27891,6	319,433	515588	190967	20002,1
106582	560,531	780333	140500	28640,4	308,372	516084	143155	20155,4
106647	635,707	780571	210151	28449,2	318,42	516335	184656	19078,7
106685	603,299	780270	194519	27732,8	312,954	515885	177153	19273,8
106747	537,453	779496	171799	25677,1	349,041	515863	195981	19613,3
106789	596,339	779076	157555	26860,2	304,9	515214	175615	19160,1
106785	735,275	779144	288040	27302,5	307,156	515225	169835	19114,9
106784	602,881	779144	152972	27488,7	349,207	519102	191193	21083,5
106789	551,889	780278	186605	25989,8	330,312	516272	157679	21654,4
106791	629,879	779911	188261	27044,5	331,702	516130	210326	22550
106800	524,302	779454	123557	28268,1	371,482	517950	189574	20914,4
106810	455,909	781262	101683	27230,5	327,517	520238	206888	19242,6
106811	721,118	815572	192264	30803,8	311,755	558579	169176	19104,2
106812	725,967	835499	230075	30548,7	416,688	555817	213746	21275,6
106811	656,659	831821	234233	29105,8	263,419	538241	128506	17605,5
106809	674,326	795126	169168	28169,2	331,885	525293	172991	20063,7
106807	783,326	838572	208506	28603	358,037	543361	215501	19475,4
106809	757,467	816116	254767	29454,1	301,015	540297	157367	17839,6
106810	746,67	831627	188868	29473,8	356,384	525847	201232	19824,5
106804	884,624	806818	202259	27584,1	264,282	564767	163456	16708,8
106800	694,239	811746	180781	28239,5	218,712	539578	95597,4	17540,9
106798	597,015	839488	168120	28588	392,069	560142	222087	21334,4

Tabulka A.2: Tabulka časy pro budovu Blox

	JPS+(0)	JPS+B(0)	A*(0)	D(0)	JPS+(1)	JPS+B(1)	A*(1)	D(1)
9166	108,117	14198,5	3119,43	3393,09	84,172	12186,4	4729,7	3231,15
9882	58,675	14760,7	3125,32	3683,25	53,184	12689,9	6132,26	3162,16
11731	59,969	18337,7	2179,66	3788,31	49,982	16011,6	1931,37	3571,92
15927	92,905	28736,7	8889	4910,56	67,881	24479,5	10997,2	4632,03
16169	108,844	29000,5	15767,1	5230,95	72,444	24787,2	22188,5	4120,06
16409	98,658	29659,1	14199,6	5290,29	71,285	25139,5	12882,9	4792,65
16923	114,701	31139,2	11933,5	5230,59	57,205	26250,1	6446,08	4651,2
17108	104,278	31515,6	10088,2	5304,34	65,906	26539,5	12669,1	4342,14
17352	136,212	32318,4	16889	5537,17	94,408	26908,7	25390,2	4542,02
17417	128,196	32547,8	18255,1	5310,82	78,802	27186,5	21378,5	4622,54
17607	153,795	33099,3	22091,3	5626,99	77,496	27544,3	23370,2	4197,41
17825	113,982	33609,5	12329	5493,66	75,565	28390,4	12897,9	5224,42
18171	108,349	34544,7	10537,6	5580,41	78,787	29155,6	14284,4	5342,67
18552	131,107	35469,4	15313	5770,27	68,495	30045,8	9214,42	5027,24
19668	168,122	38622,9	19825,8	5708,64	97,492	32804,3	14033,5	5358,79
21919	102,877	45457	8308,13	5774,75	87,62	38754,3	10671,7	4493,64
22397	155,929	47104,7	20053	6073,47	116,6	39889,9	17153,1	5007,48
24468	105,618	53996,6	8133,75	5953,9	135,929	45376,9	15150,9	5338,11
27266	146,009	64105,8	12793,3	6037,01	131,725	49364,5	25174,2	5714,64
29617	216,355	65555	38264,8	8321,49	120,37	56998,3	22724,9	4922,52
31432	237,61	73321,9	31055,3	7645,85	140,302	63840,5	23411,9	6142,67
32746	204,593	85955,8	24343,1	7719,59	135,257	69813,2	21936,5	6415,5
33215	140,485	87959,9	9852,33	7266,56	148,884	72328,1	26041,6	5976,27
34143	207,208	92097,6	17404,9	7474,66	204,616	74935,1	33574,4	6714,02
35708	262,948	97162,4	24291,1	8408,64	204,364	80008,5	38592,3	7533,4
37081	285,999	102991	38712,5	9189,09	142,567	75344	30574	6893,14
38514	272,92	91116,9	39749	9723,1	154,842	78413,6	27918,4	5198,48
39119	246,274	94894,2	36007,7	9148,8	178,416	81507,9	34392,9	5995,35
42082	287,997	126315	33206,9	9627,29	203,491	108043	35399,8	7894,1
45496	277,301	145190	32469,5	11026	180,418	124587	21267,3	8975,07
49606	430,452	169734	69031,5	12972,2	253,606	144870	58499,8	10558,6
50348	385,472	188278	62004,3	13463	187,246	147102	47380,4	11555
50367	308,621	175922	56607,2	12569,1	201,449	137697	73912,8	9088,75
50430	346,311	184714	73152,9	13309,3	174,289	143566	74780	9893,58
50412	390,632	185397	90447,3	13664,8	149,735	137849	55573	9130,83
50393	345,424	183920	84110	12265,9	198,138	137919	69143,2	9589,12
50382	378,199	163382	99475,9	12666,4	173,5	136694	60456,2	9314,25
50486	302,038	163166	63398,3	13466,9	187,301	136569	68563,7	8325,45
50472	347,342	163261	79070,1	13516,3	167,899	136444	66849,4	8191,47
50613	401,168	164726	93933,3	13193,1	176,355	136934	78496,4	9280,11
50610	349,143	164481	85044,5	13082,2	163,663	136881	63329,3	8474,18
50595	301,029	164371	82297,4	12752,2	182,934	136958	80917,8	9181,94
50586	315,573	164629	85396,2	11374,1	156,593	136574	53983,2	8638,35
50584	357,682	164312	94562,2	12816	158,248	137150	70043,4	8313,29
50586	312,256	165265	78913	12960,3	158,903	136775	68344,3	8504,32
50644	295,483	164583	75241	11255,5	170,519	137762	63176	8403,97
51666	315,44	170624	83054	13390,9	191,568	143684	80658,9	9082,04
53696	315,775	181545	82037,8	12430,8	214,241	152908	69944,1	9501,99
58132	330,118	227649	65536,8	12405,1	254,604	182985	63689	10758
73877	504,465	362526	103300	16561	366,621	298712	116573	13315,9
83604	468,418	465751	148005	19752,9	243,266	381435	123033	13664,5
85712	406,295	483745	136718	19801,1	262,138	405964	144134	14936,7
88121	466,815	514879	149000	19888,1	229,011	427954	149226	15457,8

	JPS+(0)	JPS+B(0)	A*(0)	D(0)	JPS+(1)	JPS+B(1)	A*(1)	D(1)
89375	402,264	536071	148917	20396,8	230,409	445005	168251	14999,9
92444	369,734	565294	191482	21068,3	235,989	484342	194390	16119,2
95683	346,43	606767	135269	21468,6	328,214	509900	244877	19737,2
101167	491,91	681744	206769	22684,8	312,45	563958	206759	17886,3
104400	352,115	717244	227136	22955,5	243,101	602285	198525	17467,1
105941	515,63	741918	258390	26194,8	282,607	614586	216594	19397
107606	541,966	759534	275365	23320,3	275,604	632677	202315	17587,6
108494	549,819	769676	228064	25266,8	316,606	661294	210019	19029,4
110349	564,742	818100	289073	26462,1	369,251	675838	261422	19953,5

ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Zelený** Jméno: **Václav** Osobní číslo: **434932**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra řídicí techniky**

Studijní program: **Kybernetika a robotika**

Studijní obor: **Systémy a řízení**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Aplikace rychlých plánovacích algoritmů v úloze prohledávání neznámého prostoru

Název bakalářské práce anglicky:

Application of fast planning algorithms in the exploration task

Pokyny pro vypracování:

1. Seznamte se s rychlími algoritmy plánování na mřížkách.
2. Spolu s vedoucím práce vyberte vhodnou metodu a tuto implementujte, případně nalezněte její open source implementaci.
3. Upravte Vaši implementaci tak, aby pracovala s mřížkami obsazenosti, které generuje knihovna gmapping v ROS.
4. Výsledný kód použijte v softwarovém rámci pro prohledávání neznámého prostoru mobilním robotem.
5. Proveďte experimentální ověření vlastností realizovaného algoritmu v simulovaném prostředí.

Seznam doporučené literatury:

[1] Daniel Harabor and Alban Grastien. 2014. Improving jump point search. In Proceedings of the Twenty-Fourth International Conferenc on International Conference on Automated Planning and Scheduling (ICAPS'14), Steve Chien, Minh Do, Alan Fern, and Wheeler Ruml (Eds.). AAAI Press 128-135.
[2] Daniel Harabor and Alban Grastien. 2011. Online graph pruning for pathfinding on grid maps. In Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI'11). AAAI Press 1114-1119.
[3] Uras, Tansel, and Sven Koenig. "Identifying hierarchies for fast optimal search." In Seventh Annual Symposium on Combinatorial Search. 2014.
[4] Sturtevant, N.R., Traish, J., Tulip, J., Uras, T., Koenig, S., Strasser, B., Botea, A., Harabor, D. and Rabin, S., 2015, May. The grid-based path planning competition: 2014 entries and results. In Eighth Annual Symposium on Combinatorial Search.
[5] Traish, J., Tulip, J. and Moore, W., 2016. Optimization Using Boundary Lookup Jump Point Search. IEEE Transactions on Computational Intelligence and AI in Games, 8(3), pp.268-277.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

RNDr. Miroslav Kulich, Ph.D., CIIRC

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **02.10.2017** Termín odevzdání bakalářské práce: **09.01.2018**

Platnost zadání bakalářské práce: **30.06.2019**

RNDr. Miroslav Kulich, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Ing. Pavel Ripka, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

