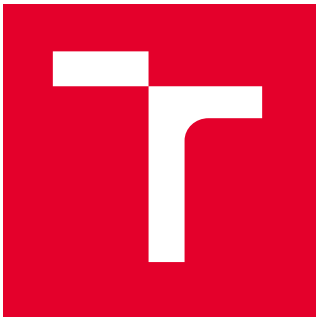


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY

INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

SUBOPTIMÁLNÍ METODY PLÁNOVANÍ CESTY

SUBOPTIMAL METHODS FOR PATH PLANNING

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. Patrik Rybníček**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Petr Šoustek**

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Patrik Rybníček**
Studijní program:	Aplikovaná informatika a řízení
Studijní obor:	bez specializace
Vedoucí práce:	**Ing. Petr Šoustek**
Akademický rok:	2022/23

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Suboptimální metody plánovaní cesty

Stručná charakteristika problematiky úkolu:

Úkolem systému pro plánování cesty je najít cestu z počáteční do cílové pozice tak, aby se agent nedostal do kolize se známými překážkami. V případě rozsáhlého stavového prostoru je obvykle nalezení optimální cesty velmi časově náročné, avšak i malá odchylka od optimální cesty může značně snížit čas potřebný k nalezení dobrého řešení. Z tohoto důvodu vznikla řada suboptimálních metod, které se úspěšně využívají např. v robotice nebo ve videohrách.

Cíle diplomové práce:

Analyzovat přístupy k plánování cesty.
Implementovat vybrané suboptimální metody plánování cesty.
Provést a vyhodnotit ověřovací a srovnávací experimenty.

Seznam doporučené literatury:

CHEN, Jingwei, et al. Revisiting suboptimal search. In: Twelfth Annual Symposium on Combinatorial Search. 2019.

CHEN, Jingwei; STURTEVANT, Nathan R. Necessary and Sufficient Conditions for Avoiding Reopenings in Best First Suboptimal Search with General Bounding Functions. In: Proceedings of the AAAI Conference on Artificial Intelligence. 2021. p. 3688-3696.

EDELKAMP, Stefan a Stefan SCHRODL. Heuristic search: theory and applications. Amsterdam ; Boston: Morgan Kaufmann, 2012. ISBN 9780123725127.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2022/23

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D. děkan fakulty

ABSTRAKT

Tato práce se věnuje metodám pro plánování cesty, konkrétně suboptimálním metodám. Ty nemusí nalézt optimální trasu, což je vyváženo nižšími nároky na paměť a čas. V teoretické části se práce zabývá problematikou hledání cesty a popisem vlastností jednotlivých metod. Praktická část implementuje sadu suboptimálních metod v simulačním prostředí vytvořeném v jazyce Python. Tyto metody následně srovnává a vyhodnocuje jejich efektivitu.

ABSTRACT

This thesis focuses on methods for path planning, specifically suboptimal methods. These may not find the optimal route, which is balanced by lower memory and time requirements. The theoretical part of the thesis deals with the problem of path finding and describes the properties of each method. The practical part implements a set of suboptimal methods in a simulation environment developed in Python. It then compares these methods and evaluates their effectiveness.

KLÍČOVÁ SLOVA

Plánování cesty, suboptimální metody, algoritmus A*, algoritmus wA*.

KEYWORDS

Path planning, suboptimal methods, A* algorithm, wA* algorithm.

ÚSTAV AUTOMATIZACE A INFORMATIKY

2023

BIBLIOGRAFICKÁ CITACE

RYBNÍČEK, Patrik. *Suboptimální metody plánovaní cesty*. Brno, 2023. Dostupné také z: `https://www.vut.cz/studenti/zav-prace/detail/149817`. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, Vedoucí práce: Ing. Petr Šoustek.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato diplomová práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury.

Jako autor uvedené práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků.

V Brně dne 26. 5. 2023 .....

Patrik Rybníček

PODĚKOVÁNÍ

Rád bych poděkoval svému vedoucímu práce Ing. Petru Šoustkovi za všechny cenné rady, připomínky, ale zejména pak ochotu a vstřícnost při psaní diplomové práce. Děkuji také své rodině za podporu a trpělivost po celou dobu mého studia.

OBSAH

1 ÚVOD

Metody plánování cest zasahují do celé řady odvětví. V průmyslu se využívá těchto metod pro plánování cest robotů či automaticky řízených vozidel. Velké uplatnění nacházejí ve videohrách, ať už se jedná o plánování cest jednotek ve strategiích nebo jedince při navigaci na herní mapě. Obecně lze říct, že videohry mají na algoritmy plánující cestu objektu stále vyšší paměťové i výkonnostní nároky. V kontextu strojní výroby se plánování cesty, respektive dráhy, používá u robotických manipulátorů. Často se jedná o jednoduchý pohyb z bodu A do bodu B, avšak do úvahy se musí brát okolní překážky a vlivy.

Pro plánování cesty jsou využívány algoritmy umělé inteligence, které lze dále dělit na několik podkategorií. Jednou z nich jsou tzv. suboptimální metody, které nezaručují optimalitu nalezené cesty, tedy nalezení nejkratší cesty ze startu do cíle, ale díky tomu dokáží prostor prohledávat mnohem efektivněji s mnohem nižšími časovými i paměťovými nároky.

Cílem této diplomové práce je popis problematiky plánování cest v kontextu suboptimálních metod. Nejprve budou uvedeny klasické prohledávací metody, na které návážou metody suboptimální. Dále by měla práce vybrané suboptimální metody implementovat a provést ověřovací a srovnávací experimenty.

2 PLÁNOVÁNÍ CESTY

Plánování cesty spočívá ve vyhledávání co nejkvalitnější trasy v prostoru. Hledání v prostoru je důležitou součástí algoritmů od jejich počátku, kdy se staly nástrojem pro řešení mnoha problémů. Od té doby se objevilo mnoho vyhledávacích algoritmů, včetně těch nejnovějších v oblasti akčního plánování, robotiky i výpočetní biologie [9].

Mezi ketegorie plánování cesty lze zařadit tzv. subtraktivní plánování cesty (plánování cesty nástrojů ve 3D prostoru v průběhu opracovávání výrobku - odebírání materiálu) a aditivní plánování cesty (výpočet dráhy nástroje, jež lze zredukovat z 3D na 2D prostor). Tato práce se však zabývá výhradně třetím druhem plánování cesty, které může být označeno jako plánování cesty robotu, konkrétněji plánování cesty v mřížce. Robotické plánování dráhy namísto odebírání nebo přidávání materiálu při výrobě objektu vypočítává, jak se může objekt pohybovat v prostoru se známými nebo neznámými překážkami a zároveň minimalizovat kolize [1]. Tento problém lze definovat jako hledání nejkratší cesty.

2.1 Popis problému prohledávání

Jak již bylo zmíněno, plánování trasy, často s podmínkou nalezení nejkratší cesty mezi dvěma body, je důležitou aplikační oblastí vyhledávacích algoritmů. Jako základní reprezentaci prostoru, ve kterém dochází k prohledávání, si lze představit graf $G = (V, E)$, v němž uzly V představují křižovatky a hrany E spojení mezi uzly (viz obr. 1). Více o reprezentaci prostředí v následující části. Úloha spočívá v nalezení cesty mezi výchozím a cílovým místem, jež jsou vstupními parametry daného problému. Algoritmus poté minimalizuje vzdálenost takové cesty, její očekávaný čas či další související míry. Tento problém může být v praxi náročný vzhledem k velkému množství uzlů a cest, které je třeba uložit do paměti, a k časovým omezením, např. navigačního systému [9].

Výsledkem hledání je poté posloupnost pozic či akcí agenta, které musí podniknout či projít, aby se ze startovní pozice dostal do cíle. Přitom se musí vyhnout všem překážkám daného prostředí. Ve složitějším případě lze prostředí považovat za dynamické, což znamená, že se překážky mohou pohybovat a nebo se mění pozice ostatních agentů, které se tak pro ostatní agenty stávají nedostupnými. Tato práce se zaobírá algoritmy, které předpokládají znalost celého prostředí. Prostředí je uvažováno jako statické a velikost procházejícího agenta je ideální, tedy nulová.

Do prvotní úvahy spadá kromě vstupu a výstupu i funkce následnovníka, tedy funkce, která umožňuje rozhodovat o směru prohledávání daného prostoru.

Obr. 1: Příklad problému plánování trasy, převzato z [9]

V neposlední řadě je důležité vzít do úvahy všechna optimalizační kritéria, jako nalezení nejkratší cesty či minimalizace časové nebo cenové náročnosti [5].

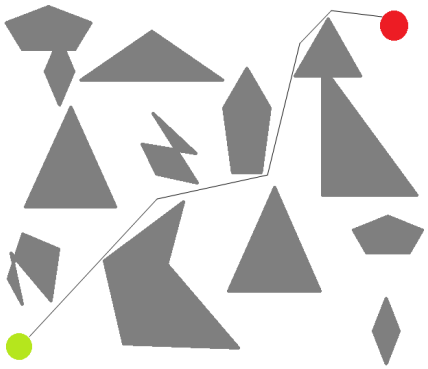
2.2 Reprezentace prostředí

Metody plánování cesty jsou založeny na předzpracování geometrických informací, které se následně promítají do datové struktury, se kterou algoritmus může pracovat. Grafovou strukturu popisující prostředí problému lze vyobrazit několika způsoby. Zobrazení grafu, tedy mapa, může mít velký vliv na výkon a kvalitu cesty.

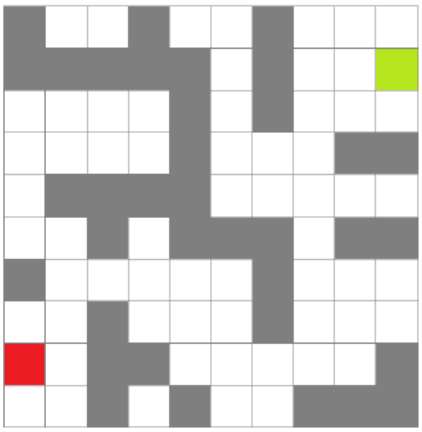
Složitost algoritmů pro hledání cesty bývá horší než lineární. To znamená, že pokud se zdvojnásobí vzdálenost potřebná k ujetí, trvá nalezení cesty více než dvakrát déle. Hledání cesty si lze představit jako prohledávání oblasti, například kruhu. Když se průměr kruhu zdvojnásobí, má čtyřikrát větší plochu. Obecně platí, že čím méně uzlů v zobrazení mapy je, tím rychlejší bude nalezení cesty [18].

Aby došlo k minimalizaci počtu uzlů, ukládají se mapy jako diskrétní aproximace reálného prostoru (mřížka nebo síť). Lze uložit i spojité aproximace map a to definováním vnitřních a vnějších hranic ve formě mnohoúhelníků a cest kolem hranic jako posloupnosti bodů s reálnou hodnotou. Přestože spojité mapy mají zjevné paměťové výhody, diskrétní mapy jsou v plánování cest nejčastější, protože se dobře graficky i vypočetně zpracovávají.

Samotnou diskretizaci je však vhodné také minimalizovat, jelikož platí, že čím více se uzly shodují s reálnými pozicemi, na kterých se agent pohybuje, tím kvalitnější výsledná cesta je [10].



Obr. 2: Spojitý vyhledávací prostor, převzato z [16]



Obr. 3: Diskretizovaný vyhledávací prostor, převzato z [16]

Mřížky diskretizují prostor na čtverce s libovolným rozlišením a každému čtverci přiřazují buď binární, nebo pravděpodobnostní hodnotu, zda je pozice obsazená, nebo prázdná. Tyto mapy lze dobře optimalizovat z hlediska paměti [1].

2.3 Metody pro plánování cesty

Po úvodním procesu vytvoření datové struktury (grafu, mapy, mřížky) následuje aplikace vyhledávacího algoritmu. Tyto algoritmy pracují systematicky tak, že postupně prozkoumávají graf nebo síť a hledají cestu spojující výchozí bod s cílovým. Základní vyhledávací algoritmy se potýkají s několika problémy, v čele s velikostí vyhledávacího prostoru. Proto je nasnadě vyhnout se opakujícímu se, respektive opětovnému procházení již prozkoumaných uzlů grafu. Nebo se vyhnout neperspektivním směrům vyhledávání.

Pro řešení těchto problémů lze vyhledávací algoritmy rozdělit do dvou kategorií: metody neinformovaného vyhledávání a metody informovaného vyhledávání.

Zatímco první zmíněné metody nevyužívají k řízení svého vyhledávání žádné informace o problémové doméně nebo stavu cíle, druhá kategorie tyto znalosti využívá, což může výrazně snížit náklady na provádění cesty jako takové a tedy zvýšit jejich efektivitu.

Rozdělení algoritmů hledajících cestu lze také uchopit z hlediska prostoru, ze kterého systém čerpá své informace. Pokud z nejbližšího okolí, jedná se o algoritmy lokální. V opačeném případě jde o globální algoritmy. Dalším odlišujícím atributem může být i míra jejich systematičnosti. Pokud je alespoň část prohledávacího procesu založena na náhodě, lze algoritmus označit jako stochastický. V opačném případě jde pak o algortimus systematický, kterým se tato práce zaobírá.

Pochopení silných stránek a omezení těchto základních vyhledávacích algoritmů je klíčové pro návrh efektivních řešení pro hledání cesty v nejrůznějších oblastech [25, 14, 12].

2.3.1 Neinformované metody

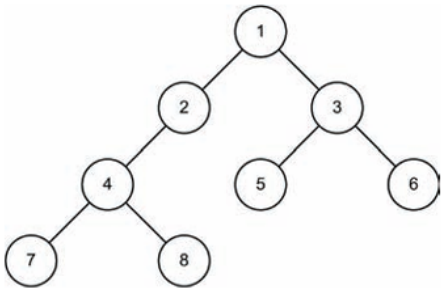
Neinformované metody jsou skupinou algoritmů, které provádějí své operace hrubou silou a kromě znalosti bezprostředního okolí daného uzlu nemají žádné další informace o vyhledávání. Jinými slovy se dají tyto algoritmy nazvat jako slepé nebo naivní vyhledávací algoritmy. Tyto algoritmy lze použít na různé vyhledávací problémy, ale protože neberou v úvahu cílový problém, jsou neefektivní. Pro jednoduchost jsou následující metody vysvětleny na stromové struktuře. Dají se však přenést i do jakékoliv další reprezentace pomocí jiných grafů či mřížky [14, 21].

Breadth-first search

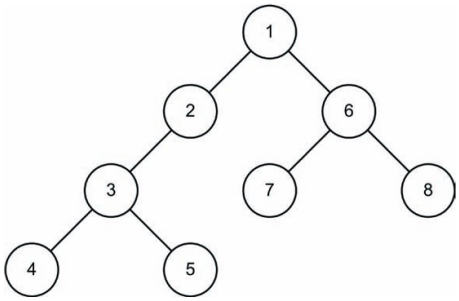
Breadth-first search (BFS) neboli vyhledávaní do šířky je jednoduchý algoritmus, při kterém se nejprve rozšiřuje hlavní uzel a až poté se rozšiřují všichni jeho nástupci, následně nástupci nástupců a tak dále (viz obr. 4). Obecně se všechny uzly rozšiřují do určité stejné hloubky v rámci vyhledávacího stromu, než se rozšíří uzel na další úrovni. Implementace algoritmu BFS používá frontu FIFO (first in, first out) neboli první dovnitř, první ven. Jakmile jsou nalezeny nové uzly, které je třeba prohledat, jsou tyto uzly porovnány s cílem, a pokud cíl není nalezen, jsou nové uzly přidány do fronty. Pro pokračování v hledání je nejstarší uzel vyřazen z fronty.

Depth-first search

Algoritmus Depth-first Search (DFS) neboli vyhledávání do hloubky je technika prohledávání grafu, která začíná v kořenovém uzlu a vyčerpávajícím způsobem prohledává každou větev do největší hloubky, než se vrátí k dříve neprozkoumaným větvím (viz obr. 5). Nalezené, ale dosud neprohledané uzly jsou uloženy ve frontě LIFO (last in, first out) známé také jako zásobník.



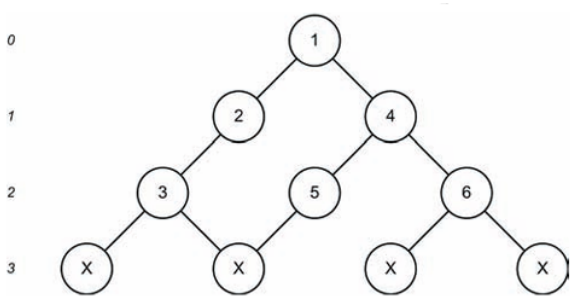
Obr. 4: Pořadí vyhledávání algoritmu BFS, převzato z [14]



Obr. 5: Pořadí vyhledávání algoritmu BFS, převzato z [14]

Depth-limited search

Depth-limited search (DLS) neboli algoritmus hledání s omezením hloubky je modifikací prohledávání do hloubky, která omezuje hloubku, do které může algoritmus vstoupit. Kromě toho, že se začíná s kořenovým a cílovým uzlem, je jako vstupní parametr stanovena i hloubka, pod kterou algoritmus nesestoupí. Všechny uzly pod touto hloubkou jsou z vyhledávání vynechány. Hledání se zastaví při dosažení této hloubky.

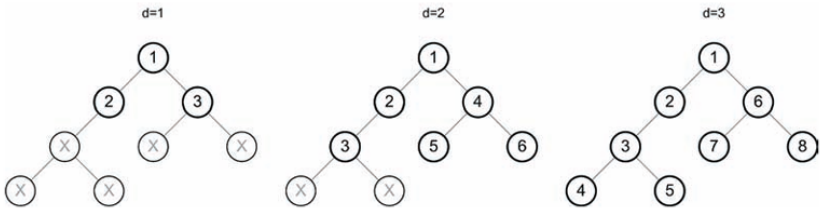


Obr. 6: Pořadí vyhledávání algoritmu DLS (hloubka = 2), převzato z [14]

Iterative deepening search

Iterative deepening search (IDS) neboli iterativní prohlubující vyhledávání kombinuje vlastnosti prohledávání do hloubky s prohledáváním do šířky. Funguje tak, že

provádí prohledávání DLS s rostoucí hloubkou, dokud není nalezen cíl nebo dokud nelze vypsat další uzly. Minimalizací hloubky prohledávání je algoritmus nucen prohledávat také šířku grafu. Pokud není cíl nalezen, hloubka, kterou smí algoritmus prohledávat, se zvýší a algoritmus se spustí znovu.



Obr. 7: Iterační prohledávání pomocí IDS, převzato z [14]

2.3.2 Důležité pojmy

Před představením druhé kategorie základních metod vyhledávání budou uvedeny vlastnosti algoritmů, které jsou důležité pro další popis a implementaci jich samotných i následně představených algortimů.

Výpočetní složitost

Základním měřítkem efektivity algoritmu je výpočetní složitost. Tu můžeme rozdělit na časovou, pokud zjišťujeme čas, za který algoritmus dojde k řešení, a prostorovou. Prostorovou změříme na základě paměťové náročnosti, která roste s velikostí prohledaného prostoru. Celkově pak mluvíme o množství prostředků, které jsou potřeba k jeho provozu. Pro popis složitosti algoritmů se používá tzv. Big-O notace (viz tab. 1), jež definuje asymptotickou horní hranici algoritmu jako horní hranici odhadu složitosti. Zápis Big-O poskytuje míru složitosti vyhledávacího algoritmu v nejhorším případě [6].

Tab. 1: Big-O notace vzhledem k výpočetní složitosti, převzato z [14]

Big-O notation	**Složitost**
$O(1)$	Konstatní
$O(n)$	Lineární
$O(log(n))$	Logaritmická
$O(n^2)$	Kvadratická
$O(c^n)$	Geometrická
$O(n!)$	Kombinatorická

Úplnost

Vedle složitosti se u algoritmu řeší, zda je úplný. Algoritmus je úplný právě tehdy, když najde řešení (v tomto případě cestu mezi dvěma body), pokud nějaké existuje. Neúplný algoritmus toto řešení najít nemusí, a to ani ve chvíli, kdy existuje. Úplnost je důležitá vlastnost, kterou je třeba zvážit při výběru algoritmu pro daný problém. Pokud je známo, že problém má řešení, a je důležité toto řešení najít, musí být zvolen úplný algoritmus. Úplnost však často přichází za cenu zvýšené výpočetní složitosti, takže neúplné algoritmy mohou být upřednostňovány v situacích, kdy je rychlost nebo efektivita důležitější než úplnost. Zatímco pro algoritmus BFS lze úplnost garantovat v jakémkoliv případě, pro DFS úplnost není obecně zaručena. Lze říci, že algoritmus hledání do hloubky je úplný, pokud je prohledávaný prostor konečný [12].

Schopnost ukončení

Ukončení algoritmu je proces, kdy algoritmus zastaví a vrátí uživateli výsledek. Když je spuštěn s určitými vstupními požadavky, nakonec se zastaví a vytvoří výstup, a to buď úspěšně nebo s chybou. Lze říci, že ukončení je i důležitou vlastností algoritmů, protože zajišťuje, že je lze spolehlivě použít k řešení problémů. Proces, který se neukončí nebo dává nesprávný výstup, není v praxi použitelný a nelze ho pojmenovat jako algoritmus. Proto je zajištění správného ukončení klíčovým aspektem při návrhu a testování algoritmů [12].

2.3.3 Informované metody

Zatímco neinformované metody prohledávají prostor sice systematicky, ale hrubou silou, dále představené informované metody jsou třídou agoritmů, které k řízení procesu používají informace o svém lokálním i globálním okolí. Někdy se tyto metody nazývají heuristické (více o heuristikách v další části). Zjednodušeně řečeno, informované algoritmy se rozhodují v každém kroku o dalším pokračování nebo směřování dle kvality svého okolí. Ať už jde o již prozkoumanou oblast, tak i o odhad zatím neexpandované oblasti [14].

Best-first search

Informované algoritmy jsou často efektivnější než neinformované algoritmy, protože upřednostňují průzkum nejslibnějších cest. Základní algoritmus informovaných metod se nazývá Best-first search (BestFS), při kterém je následný uzel pro rozšíření v grafu vybrán na základě evaluační funkce $f(n)$, respektive je formou této funkce vyhodnocen nejlepší adept ze všech aktuálních adeptů.

Modifikací tohoto algoritmu je Greedy best-first search (GBFS), který používá heuristickou funkci k odhadu vzdálenosti nebo nákladů na cestu do cílového stavu, ale nebere v úvahu náklady na dosud prošlou cestu:

$$f(n) = h(n) \tag{1}$$

Greedy best-first search zkoumá cestu, která se na základě heuristického odhadu jeví jako nejslibnější, aniž by bral v úvahu skutečné náklady na cestu. To může vést k neoptimálním výsledkům, ale algoritmus je v mnoha případech velmi rychlý, protože prochází méně uzlů [25].

Informované algoritmy jsou kvalitním nástrojem pro problémy hledání cest a při správném použití zvyšují efektivitu procesu hledání. Velký faktor v této věci hraje kvalita heuristické funkce, kterou algoritmus používá. Špatná nebo nesprávně použitá heuristika může vést k neoptimálním výsledkům nebo dokonce k nenalezení řešení.

Termín Best-first search používají různí autoři trochu jiným způsobem. Russellova a Norvigova učebnice [25] nazývá Best-first search jakýkoli algoritmus, který seřadí uzly podle nějaké funkce a pokaždé vybere ten nejlepší. Judea Pearl má poněkud odlišnou definici Best-first [19]. Hlavní rozdíl je ve specifikaci, jak zacházet s již expandovanými uzly (Russell a Norvig nespecifikují, co dělat), a v tom, kdy se provádí test cíle (Russell a Norvig v Graph-Search kontrolují cíl pouze na prvním prvku OPEN, Pearl kontroluje, jakmile jsou generováni následníci uzlu) [12].

Pro lepší představu je třeba doplnit slovník o více pojmů. Nejprve prioritní fronta OPEN. Tato fronta schraňuje již nalezené uzly v grafu, ale zatím neexpandované, též nerozšířené. Oproti tomu fronta CLOSED obsahuje všechny stavy, které již byly rozšířeny a dále se s nimi neplánuje pracovat. Pokud se řekne, že se expanduje následující uzel, znamená to, že se vezme nejlepší uzel (s nejnižším ohodnocením) z OPEN a přesune se do CLOSED, přičemž se všichni jeho přípustní sousedé (není to překážka či nedovolený stav) přesouvají automaticky do OPEN. Průběh BestFS vypadá následovně:

Algoritmus 1 Best-first search algorithm

```
1: function BestFS(start, goal)
2:     Put start on OPEN
3:     while OPEN not empty do
4:         Take best n from OPEN
5:         if n == goal then
6:             return path(start,goal)
7:         end if
8:         for each s in succesors(n) do
9:             if s on CLOSED then
10:                return
11:            end if
12:            if s on OPEN then
13:                if cost(s) < cost in OPEN then
14:                    update cost
15:                end if
16:            else
17:                add s to OPEN
18:            end if
19:        end for
20:    end while
21:    return no path
22: end function
```

Heuristiky

Heuristika je kritérium, metoda nebo princip pro rozhodování, který pomáhá nálézt z několika alternativních způsobů nejefektivnější směr pro dosažení určitého cíle. Představuje kompromis mezi dvěma požadavky: potřebou, aby toto kritérium bylo jednoduché, a zároveň přáním, aby správně rozlišovalo mezi dobrými a špatnými možnostmi.

Z hlediska metod pro plánování cesty je heuristika určená k rychlejšímu řešení problému, pokud jsou neinformované metody příliš pomalé pro nalezení řešení nebo když nenajdou řešení žádné. V případě matematického popisu jde o heuristickou funkci, která v prohledávacích algoritmech v každém kroku rozhodování o směru dalšího postupu řadí alternativy na základě dostupných informací a rozhoduje, který směr následovat. Může například aproximovat přesné řešení (vzdálenost k němu) a díky tomu zvýšit rychlost nalezení cíle. Toho dosahuje výměnou za optimalitu (probráno v další části), úplnost či přesnost.

V problematice hledání cesty je heuristická funkce odhadem minimálních nákladů pro cestu z libovolného místa v prostoru do cíle. Jinak řečeno, jedná se nikoliv o reálnou vzdálenost, ale o odhad vzdálenosti z aktuální pozice vzhledem k cíli. Pro hledání cesty v prostoru reprezentovaným mřížkou jsou nejčastěji používané Manhattanská metrika, Euklidovská metrika, diagonální a Čebyševova metrika [19, 17].

Manhattanská metrika udává vzdálenost mezi dvěma body a je měřena podél os svírajících pravý úhel. Vychází z pravoúhlého rozložení ulic inspirovaného jednou z částí města New York v USA. Tato metrika je vhodná pouze pro pohyb ve 4 směrech, tedy rovnoběžně s osou x a y v kartézských souřadnicích. Podoba této metriky může vypadat následovně:

Algoritmus 2 Manhattanská metrika

```
1: function MANHATTAN(node, goal):        ▷ aktuální pozice, cílová pozice
2:     dx = abs(node.x − goal.x)
3:     dy = abs(node.y − goal.y)
4:     return dx + dy                      ▷ Manhattanská vzdálenost
5: end function
```

V grafickém zobrazení pak lze vidět, že tato metrika počítá pouze s kolmými směry pohybu (viz obr. 8).

Euklidovská metrika oproti Manhattanské pracuje se všemi směry (viz obr. 8).

Algoritmus 3 Euklidovská metrika

```
1: function EUCLID(node, goal):                ▷ aktuální pozice, cílová pozice
2:     dx = abs(node.x − goal.x)
3:     dy = abs(node.y − goal.y)
4:     return sqrt(dx · dx + dy · dy)          ▷ Euklidovská vzdálenost
5: end function
```

Tzv. diagonální metrika, v angličtině taktéž označovaná jako Octile předpokládá pohyb v 8 směrech, tedy ve 4 kolmých a k tomu 4 diagonálních.

Algoritmus 4 Diagonální metrika

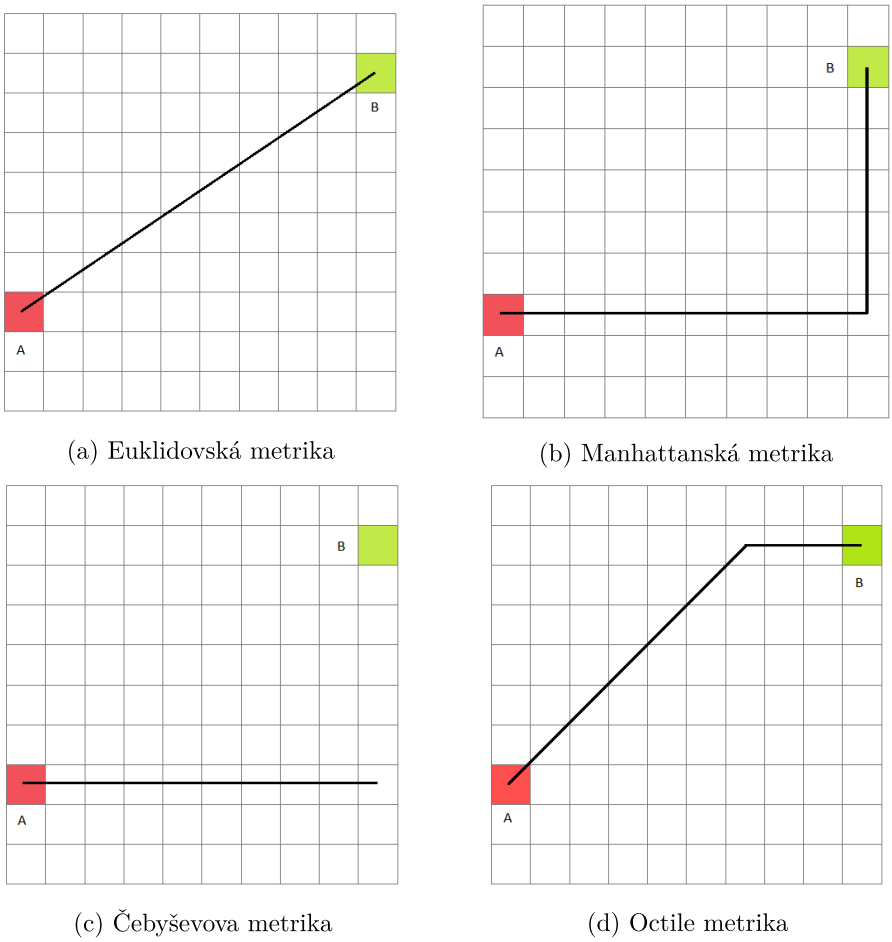
```
1: function OCTILE(node, goal):                ▷ aktuální pozice, cílová pozice
2:     dx = abs(node.x − goal.x)
3:     dy = abs(node.y − goal.y)
4:     D = sqrt(2)
5:     return dx + dy + (D − 2) · min(dx, dy)  ▷ Diagonální vzdálenost
6: end function
```

Je poznat, že grafická forma (viz obr. 8) se v mnohém od Euklidovské metriky neliší, ovšem z hlediska prohledávání je právě diagonální způsob o dost efektivnější.

Čebyševova metrika je implemetačně velmi podobná diagonální, ovšem představuje pouze vzdálenost mezi dvěma body jako největší z absolutních hodnot dvou vzdáleností podél os (viz algoritmus 5).

Algoritmus 5 Čebyševova metrika

```
1: function CHEBYSHEV(node, goal):             ▷ aktuální pozice, cílová pozice
2:     dx = abs(node.x − goal.x)
3:     dy = abs(node.y − goal.y)
4:     D = 1
5:     return dx + dy + (D − 2) · min(dx, dy)  ▷ Čebyševova vzdálenost
6: end function
```



(a) Euklidovská metrika

(b) Manhattanská metrika

(c) Čebyševova metrika

(d) Octile metrika

Obr. 8: Grafická forma heuristických metrik převzato z [16]

Přípustnost

Přípustnost je jednou z důležitých vlastností heuristiky. Přípustná heuristika je taková, která nikdy nepřeceňuje náklady na dosažení cíle. Používá se k odhadu nákladů na dosažení cílového stavu v informovaném vyhledávacím algoritmu. Přípustné heuristiky jsou ze své podstaty optimistické, protože se domnívají, že náklady na řešení problému jsou menší, než ve skutečnosti jsou. Příkladem přípustné heuristiky je přímá vzdálenost. Přímá vzdálenost je přípustná, protože nejkratší cesta mezi dvěma body je úsečka, která ze své podstaty nemůže být nadhodnocena [25].

Lépe řečeno, heuristická funkce $h(n)$ je přípustná, pokud splňuje následující podmínku pro libovolný uzel n ve vyhledávacím stromu či mřížce, ve kterém jsou hrany, respektive cesty mezi uzly, ohodnoceny nezápornou hodnotou.

$$h(n) \leq h^*(n) \tag{2}$$

kde $h^*(n)$ jsou skutečné náklady na dosažení cílového stavu z uzlu n. To znamená, že heuristická hodnota je vždy menší nebo rovna skutečným nákladům na dosažení cílového stavu.

Použití přípustné heuristiky v algoritmu pro hledání cesty je první podmínkou, aby algoritmus nalezl optimální řešení, tj. cestu s nejnižšími náklady z výchozího do cílového stavu. Je tomu tak proto, že přípustná heuristika nikdy nepřeceňuje náklady na dosažení cílového stavu, takže algoritmus vždy nejprve prozkoumá cestu s nejnižšími odhadovanými náklady.

Mezi příklady přípustných heuristik patří Manhattanská (v případě 4 možných směrů pohybu), Euklidovská i diagonální heuristika. Tyto heuristiky se běžně používají v algoritmech pro hledání cesty, aby efektivně vedly hledání k cílovému stavu a zároveň zaručily optimální řešení (více o optimalitě v části *Optimalita a suboptimalita*). Je důležité poznamenat, že ne všechny heuristiky jsou přípustné [9].

Konzistence

Druhá podmínka se nazývá konzistence (též monotónnost). Heuristická funkce $h(n)$ je konzistentní, jestliže splňuje následující podmínku pro libovolný uzel n ve vyhledávacím prostoru:

$$h(n) \leq c(n, n') + h(n') \tag{3}$$

kde $c(n, n')$ jsou náklady na přesun z uzlu n do jeho sousedního uzlu n' a $h(n')$ je heuristická hodnota uzlu n'. Jinými slovy, odhadované náklady na dosažení cílového stavu z uzlu n jsou vždy menší nebo rovny součtu odhadovaného nákladu na dosažení cílového stavu z jeho sousedního uzlu a skutečného nákladu na přesun z uzlu n do uzlu n'.

Podmínka konzistence zaručuje, že heuristická funkce nikdy nepřeceňuje skutečné náklady na dosažení cílového stavu z žádného uzlu ve vyhledávacím stromu.

Tato vlastnost je důležitá, protože zajišťuje, že algoritmus dokáže efektivně najít optimální řešení, i když nákladová funkce není rovnoměrná nebo je prohledávací prostor velký.

Všechny z již zmíněných heuristik jsou stejně jako přípustné i konzistentní. Je však podstatné poznamenat, že zatímco všechny konzistentní heuristiky jsou zároveň přípustnými heuristikami, ne všechny přípustné heuristiky jsou konzistentní [9, 25].

Optimalita

Optimalita v algoritmech pro hledání cesty je schopnost algoritmu najít nejlepší nebo optimální cestu z daného výchozího bodu do požadovaného cíle. Nejlepší nebo optimální cesta je obvykle definována jako cesta s nejnižšími možnými náklady nebo vzdáleností mezi dvěma body. Aby mohl být algoritmus pro hledání cesty považován za optimální, musí zaručit, že vždy najde cestu s nejnižšími možnými náklady [9, 25].

Celkově je optimálnost důležitým hlediskem při návrhu a implementaci algoritmů pro hledání cesty, zejména v aplikacích, kde je nalezení nejlepší nebo optimální cesty zásadní pro úspěch.

2.3.4 Algoritmus A*

Bezkonkurenčně nejznámějším algoritmem informovaných metod je algoritmus A* (čteno A-star). Vyhodnocuje uzly kombinací $g(n)$, nákladů na dosažení uzlu, a $h(n)$, odhadu nákladů na dosažení cíle. Evaluační funkce $f(n)$ má tvar, který udává odhadované náklady na nejlevnější řešení přes uzel n:

$$f(n) = g(n) + h(n) \tag{4}$$

Tato evaluační funkce je chápána jako součet odhadu nákladů z akutálního stavu do cíle $h(n)$ a nákladů již použitých v dosavadní části cesty $g(n)$. Následně se uzel s nejnižším ohodnocením rozšiřuje jako první. Pokud se tedy snažíme najít nejlevnější řešení, je rozumné se nejprve pokusit o uzel s nejnižší hodnotou $g(n)+h(n)$. Ukazuje se, že tato strategie je velmi vhodná. Za předpokladu, že heuristická funkce $h(n)$ splňuje podmínky přípustnosti a konzistence, je hledání A* (za předpokladu existence řešení) úplné i optimální [30, 25].

A* algortimus je speciálním případem hledání BestFS algoritmu, kde se dosavadní náklady $g(n)$ uzlu n vypočítají rekurzivně přičtením nákladů $g(m)$ jeho nadřazeného uzlu m k nákladům cesty mezi těmito uzly:

$$g(n) = g(m) + c(m, n) \tag{5}$$

Předpokládáme, že náklady na cestu c jsou kladné. Počáteční uzel má náklady $g(s) = 0$. Dalším předpokladem je i to, že heuristická funkce $h(n)$ je podhodnocená

cena optimální cesty z uzlu n do cíle [12]. Nazveme-li $h^*(n)$ cenu optimální cesty, tak platí, že:

$$h(n) \leq h^*(n) \tag{6}$$

Princip algoritmu

Stejně jako u BestFS algoritmu A* používá dva typy front. Nejprve se inicializuje prioritní fronta OPEN (množina uzlů, které mají být vyhodnoceny) počátečním uzlem. Dále se vytvoří také seznam CLOSED pro sledování již vyhodnocených uzlů.

Pomocí heuristické funkce se k uzlům v OPEN odhadují náklady na nejlevnější cestu z aktuálního uzlu do cílového uzlu. Tato hodnota se nazývá heuristická hodnota $h(n)$. Zároveň jsou vypočteny i celkové náklady na cestu z výchozího uzlu do aktuálního uzlu tedy hodnota $f(n)$ dle rovnice evaluační funkce. V každém cyklu algoritmu se dále sleduje, zda je aktuální uzel cílovým. Pokud ano, našel A* optimální cestu a může ji vyhodnotit. Pro každého souseda aktuálního uzlu jsou vypočteny celkové náklady na cestu k tomuto sousedovi a pokud do nich nebyla dříve nalezena kratší cesta, jsou přidáni do seznamu OPEN. Pro každého souseda se opakuje výpočet hodnot f a h.

Pokud u aktuálního stavu byly expandovány všechny směry, odstraňuje se aktuální uzel z OPEN a přiřazuje se do seznamu CLOSED. V tuto chvíli se vybere uzel z OPEN s nejnižší hodnotou f a nastaví se jako aktuální uzel. Další kroky už se opakují, dokud není množina OPEN prázdná nebo dokud aktuální uzel není uzlem cílovým.

Tab. 2: Srovnání dosavadních algoritmů, převzato z [25]

Algoritmus	**Časová složitost**	**Prostorová složitost**	**Úplné?**	**Optimální?**
BFS	$O(b^d)$	$O(b^d)$	Ano	Ano
DFS	$O(b^m)$	$O(bm)$	Ne	Ne
DLS	$O(b^l)$	$O(bl)$	Ne	Ne
IDS	$O(b^d)$	$O(bd)$	Ano	Ano
GBFS	$O(b^d)$	$O(b^d)$	Ano	Ne
A*	$O(b^d)$	$O(b^d)$	Ano	Ano

b - faktor větvení, d - hloubka řešení, m - maximální hloubka grafu, l - limit prohledávané hloubky; vše za předpokladu existujícího konečného řešení a nezáporných vah cest mezi uzly

Algoritmus 6 A*

```
1: function A*(start, goal)
2:     OPEN ← start
3:     CLOSED ← empty
4:     cameFrom ← empty
5:     g_n[start] ← 0
6:     f_n[start] ← h(start, goal)
7:     while OPEN is not empty do
8:         current ← n from OPEN with lowest f_n
9:         if current == goal then
10:            return reconstructPath(cameFrom, current)
11:        end if
12:        remove n from OPEN
13:        for all neighbor of n do
14:            tentativeG_n ← g_n[current] + distance(current, neighbor)
15:            if tentativeG_n < g_n[neighbor] then
16:                cameFrom[neighbor] ← current
17:                g_n[neighbor] ← tentativeG_n
18:                f_n[neighbor] ← g_n[neighbor] + h(neighbor, goal)
19:                if neighbor is not in OPEN then
20:                    add neighbor to OPEN
21:                end if
22:            end if
23:        end for
24:        CLOSED ← n
25:    end while
26:    return failure
27: end function
```

3 SUBOPTIMÁLNÍ METODY

Hledání optimální cesty může být mnohdy časově i prostorově náročná operace. I proto na řadu přicházejí algoritmy či jejich alternativy, které nemusí splňovat podmínku nalezení optimální cesty, ale mohou se jí velmi přiblížit a přitom proces hledání výrazně zrychlit. V takovém případě přichází v úvahu suboptimální algoritmy, které nezaručují nalezení optimální cesty, nýbrž cesty takové, která může mít vyšší náklady. Stupeň suboptimality se obvykle měří jako poměr nákladů na cestu nalezenou algoritmem a nákladů na optimální cestu.

Pokud v řešení situace nejde čistě pouze o nalezení optimální cesty, přichází vhod výhody suboptimálních metod. Ty totiž mohou být rychlejší nebo vyžadovat méně paměti než algoritmy zaručující optimalitu.

V některých aplikacích, jako je plánování a řízení v reálném čase, může být rychlost a efektivita algoritmu důležitější než nalezení optimální cesty. Suboptimální algoritmy mohou být užitečné také tehdy, když je evaluační funkce neurčitá nebo když je prohledávací prostor velmi rozsáhlý a nalezení optimální cesty je výpočetně neproveditelné.

Jak již bylo zmíněno, nevýhodou použití suboptimálních algoritmů je, že nezaručují nalezení optimální cesty. Navíc může být v určitých případech rozdíl mezi suboptimální a optimální cestou tak markantní, že vede k neefektivnímu nebo nebezpečnému provozu.

Volba mezi použitím optimálních a suboptimálních algoritmů tak závisí na konkrétních aspektech daného problému. Jestliže je kvalita nalezené cesty jediným či hlavním cílem, je dobré se zaměřit na algoritmy zaručující optimální řešení. Pokud je důležitější rychlost a efektivita, může být vhodnější suboptimální algoritmus.

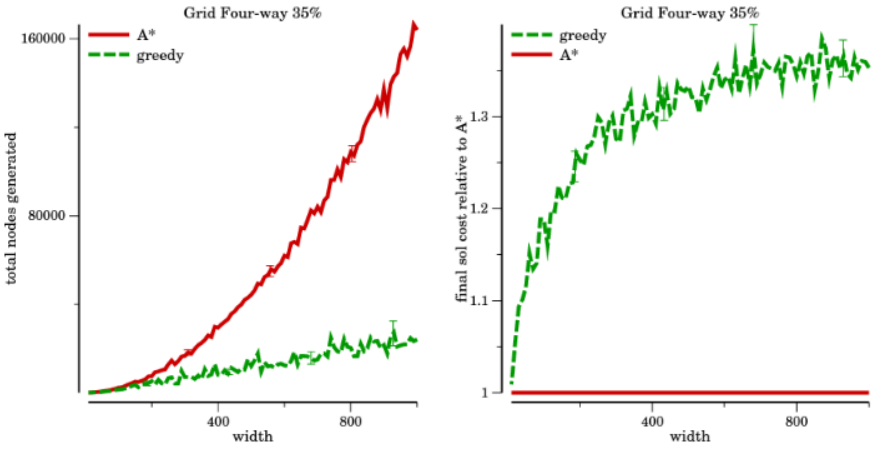
3.1 Neomezené suboptimální algoritmy

Do kategorie suboptimálních algoritmů lze zařadit již některé z algoritmů, které byly zběžně představeny. Lze je nazvat neomezené suboptimální algoritmy. Neomezené jsou zejména proto, že neposkytují žádnou záruku kvality řešení, které vytvoří. Mohou nalézt řešení velmi vzdálená od optimálních a to v závislosti na daném prostředí.

Použití najdou obvykle v situacích, kdy není nutné nalézt exaktní řešení, ale postačí rychle nalezené přibližné, tedy suboptimální. Z hlediska praktického využití lze takové chování využít například ve videohrách, kde je rychlost nalezení řešení mnohdy o dost důležitější než jeho kvalita.

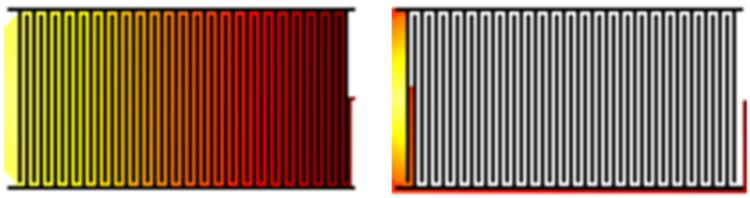
Jeden z příkladů neomezeného suboptimálního algoritmu je Depth-first search. Tento algoritmus neposkytuje žádné záruky kvality řešení, které vytvoří, protože jednoduše zkoumá cesty, dokud nedosáhne cílového stavu, aniž by bral v úvahu náklady

na cestu. Dalším příkladem je algoritmus GBFS (Greedy best-fisrt search), který zkoumá své okolí na základě heuristické funkce, ale opět neposkytuje žádné záruky kvality řešení. Na následujícím obrázku lze vidět, že i když GBFS značně předčí A* z hlediska rychlosti (množství projitých stavů), jako oběť dochází k rapidnímu poklesu kvality řešení.



Obr. 9: Srovnání algoritmů A* a GBFS, převzato z [31]

Je také vhodné si uvědomit, že praktičnost algoritmů jako je GBFS je závislá na daném prostředí. U velkých či značně komplexních map jsou výsledky GBFS oproti klasickému A* nejenom méně kvalitní, nýbrž i značně pomaleji získané (o mnoho více projitých stavů).



Obr. 10: Komplexní mapa - GBFS (vlevo), A* (vpravo), převzato z [31]

Neomezené suboptimální algoritmy mohou být v řadě případů užitečné, ale obecně se jim nevěnuje mnoho pozornosti, jelikož jsou v nejhorších případech nalezeny opravdu nekvalitní cesty, což je v praxi nežádoucí. Pokud tedy primárním cílem není co největší minimalizace časových nároků, je dobré použít omezené suboptimální algoritmy, které poskytují záruku kvality řešení (viz sekce Omezené suboptimální algoritmy) [31].

3.2 Anytime algoritmy

Další možnou kategorií suboptimálních algoritmů jsou Anytime algoritmy, které jsou navrženy podobně jako neomezené suboptimální algoritmy, a to tak, aby rychle nacházely řešení problémů, ale oproti předchozí kategorii také tak, aby dosavadní řešení ještě vylepšovaly a to v případě, že je pro algoritmus k dispozici více času. Název těchto algoritmů koresponduje s tím, že algoritmus lze kdykoli zastavit a výsledky, které byly do té doby vytvořeny, mohou být použitelné.

Algoritmy Anytime jsou tak typem suboptimálního vyhledávacího algoritmu, který lze kdykoli přerušit a který stále vrací řešení, jež se v čase zlepšuje. Řešení není nikdy garantováno jako optimální, ale samozřejmě se k němu časem může velmi přiblížit. To záleží i na tom, v jakém momentu je algoritmus ukončen.

Tyto algoritmy najdou uplatnění v řadě oblastí, ale jejich použití je vhodné pouze tehdy, když jsou výpočetní zdroje omezené v čase na základě reálných podnětů. I proto se jejich výkon či kvalita špatně srovnává s dalšími algoritmy, které se zastavují automaticky při nalezení prvního řešení. Vzhledem k tomu, že algoritmy Anytime mohou být kdykoli přerušeny, může být obtížné interpretovat srovnávací experimenty a jejich výsledky.

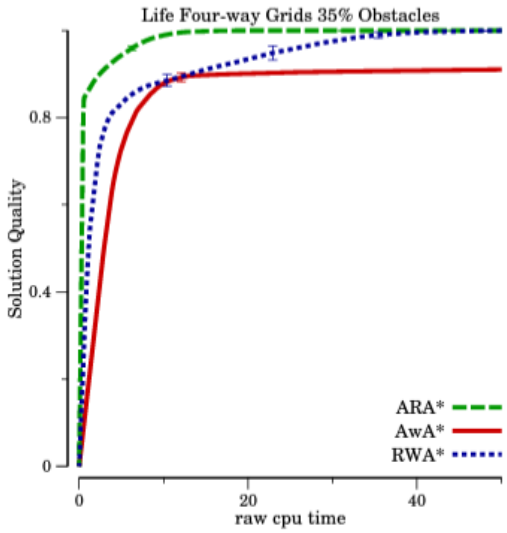
Navíc algoritmy Anytime často vyžadují více výpočetních zdrojů než další klasické algoritmy, protože musí svá řešení v čase kontinuálně zlepšovat, což může být značný problém v prostředí s omezenými zdroji. Taktéž vyžadují pečlivé nastavování parametrů, aby dosáhly kvalitních výsledků, které jsou vyváženy mezi rychlost a optimálnost.

Celkově lze říci, že ačkoli mají Anytime algoritmy své silné stránky, nejsou vhodné pro všechny problémy. Rozhodnutí o použití takového algoritmu by mělo být založeno na konkrétnách požadavcích daného problému [16, 24].

Mezi používané Anytime algoritmy patří Anytime weighted A* (AwA*) [13], Anytime repairing A* (ARA*) [15], Restarting weighted A* (RwA*) [23], které využívají principy optimálních či omezených suboptimálních algoritmů a v čase upravují jejich parametry tak, aby se postupně posunovaly k lepším výsledkům. Principielně však nejde o nijak nové algoritmy, pouze se v čase upravuje jejich nastavení. I proto se tato práce soustředí zejména na popis omezených suboptimálních metod, ze kterých tyto algoritmy čerpají [31].

3.3 Omezené suboptimální algoritmy

Asi nejpraktičtější a zároveň nejlépe měřitelnou kategorií suboptimálních algoritmů jsou metody omezených suboptimálních algoritmů. Tyto algoritmy jsou schopny samy balancovat mezi kvalitou nalezené cesty a dobou, za jakou se k tomuto řešení



Obr. 11: Srovnání algoritmů AwA*, ARA* a RwA*, převzato z [31]

dostanou. Omezení nebo podmínky kvality jsou definovány jako podíl mezi náklady řešení nalezeného suboptimálním algoritmem a náklady optimálního řešení. V praxi to tedy znamená, že tyto algoritmy dokáží zajistit kvalitu daného výsledku již jako vstupní parametr.

Omezená suboptimalita je důležitý pojem v algoritmech pro hledání cesty, protože poskytuje míru kompromisu mezi kvalitou řešení a výpočetními prostředky potřebnými k jeho nalezení. Algoritmus s menší omezenou suboptimalitou je obecně upřednostňován před algoritmem s větší suboptimalitou, protože poskytuje řešení bližší optimálnímu. Algoritmy s menší omezenou suboptimalitou však mohou k nalezení řešení vyžadovat větší výpočetní náročnost [29, 31].

3.3.1 Metody wA*

Jako základ metod omezeného suboptimálního vyhledávání je třeba uvést weighted A* neboli vážený A*. Princip této metody nastínil již v roce 1970 Ira Pohl ve svém článku *Heuristic Search Viewed as Path Finding in a Graph* [22]. Kromě celkového shrnutí důležitosti heuristického vyhledávání navrhl metodu vážení heuristické funkce, a tím započal myšlenku o vyvažování optimálního řešení vzhledem k rychlejšímu prohledávání. Tento přístup se později pojmenoval jako weighted A*, který je základním algoritmem pro suboptimální metody. Algoritmus vychází z generického Best-first search algoritmu, stejně jako A*.

Algoritmus wA*

Weighted A* je modifikací vyhledávacího algoritmu A*. Hlavním rozdílem mezi těmito algoritmy je podoba jejich evaluačních funkcí. V případě weighted A* se totiž hodnota heuristiky násobí váhou w, která určuje rozložení hodnoty mezi skutečné ná-

klady a heuristický odhad. Menší hodnota w znamená, že algoritmus upřednostňuje skutečné náklady před heuristickým odhadem, zatímco větší hodnota w znamená opak. Rovnice tohoto řešení se dá zapsat následovně:

$$f(n) = (1 - w)g(n) + wh(n) \tag{7}$$

kde n je uzel ve vyhledávacím stromu, $g(n)$ je skutečná cena cesty z počátečního uzlu do n, $h(n)$ je heuristický odhad ceny nejlevnější cesty z n do cílového uzlu a w je váhový faktor.

V případě takového zápisu můžeme uvažovat hodnoty w od nuly do jedné. Pokud by se hodnota váhy rovnala $0,5$, pak by se wA* choval stejně jako A*. Zajímavější jsou však extrémy tohoto intervalu. Pro hodnotu 1 je evaluační funkce shodná s funkcí GBFS algoritmu. Při opačné hodnotě, tedy 0, se naopak hledání promění v algoritmus Uniform-cost search, který při svém prohledávání využívá pouze informace z již expandovaných uzlů.

V praxi o něco používanější a pro účel suboptimality lepší formou je tento zápis evaluační funkce:

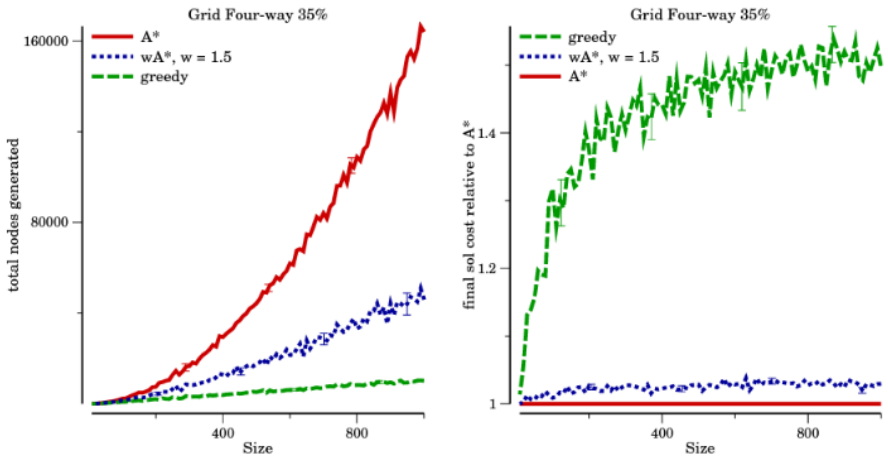
$$f(n) = g(n) + wh(n) \tag{8}$$

Weighted A* je omezeným suboptimálním algoritmem díky tomu, že lze zaručit nalezení cesty, která je nejvýše w-násobně delší než optimální cesta. To znamená, že s rostoucím váhovým faktorem w najde algoritmus méně nákladnou cestu, která však může být z pohledu nákladů až w-násobně vzdálená cestě optimální. Tato vlastnost je základním stavebním kamenem omezených suboptimálních algoritmů [22, 30, 31].

Na následujícím obrázku (viz obr. 12) je možné vidět, že samotný algoritmus má mnohem lepší výsledky oproti GBFS z pohledu kvality nalezené cesty vzhledem k počtu prohledaných či expandovaných uzlů, což je v praxi velmi cenným faktem.

Lze říct, že algoritmus wA* je úplný a w-optimální, tedy jeho výsledná cesta se nebude více než w-násobně lišit od cesty optimální. Toto tvrzení je opřeno o jednoduchou myšlenku, respektive důkaz, který se opírá o fakt, že v seznamu OPEN (viz Algoritmus A*) je vždy jeden z uzlů uzlem na optimální cestě [30]. Takový uzel ponese označení n, lze si ho představit jako předposlední uzel na cestě do cíle. Jelikož je algoritmus wA* úplný, najde vždy existující řešení, avšak nemusí najít takové, které je přímo optimální. Proto bude procházet uzlem, jež není na optimální cestě. Takový uzel ponese označení p. V následující rovnici pro jakýkoliv uzel p na výsledné cestě vytvořené algoritmem wA* platí tento důkaz o suboptimalitě:

$$f(p_i) \leq f(n) = g(n) + wh(n) \leq w(g(n) + h(n)) \tag{9}$$



Obr. 12: Srovnání algoritmů A*, GBFS a wA*, převzato z [31]

$$f(p_i) \leq wC^* \tag{10}$$

Tato nerovnice jasně dokazuje, že řešení poskytnuté wA* je vždy w-optimální. Obrázek 13 zobrazuje, že přes nalezení mírně horší cesty (než je optimální) nemusí algoritmus wA* oproti A* prohledávat tak velký prostor, respektive expandovat tolik uzlů. V tomto případě jsou uzly znázorněné zelenou barvou stavy v seznamu OPEN a červené již expandované v seznamu CLOSED.

Obr. 13: Velikost prohledaného prostoru algoritmy: A* (vlevo), wA* ($w = 2$, vpravo), převzato z [29]

Algoritmus XDP/XUP

Hlavním nástrojem obecného Best-first search algoritmu je definice tzv. nejlepšího uzlu neboli výběr uzlu, jenž se v seznamu OPEN zařadí na první místo, a stane se

tak dalším kandidátem k expandování. K definici se obvykle používá již zmíněná evaluační funkce, někdy označovaná jako prioritní. Zatímco algoritmus A* většinou zapisuje tuto rovnici ve formátu $f(n) = g(n) + h(n)$ a wA* jako $f(n) = g(n) + wh(n)$, Nathan R. Sturtevant a Jingewi Chen ji pro účely vytvoření komplexnějších evaluačních funkcí ve svém článku *Conditions for Avoiding Node Re-expansions in Bounded Suboptimal Search* [7] přepsali do tvaru:

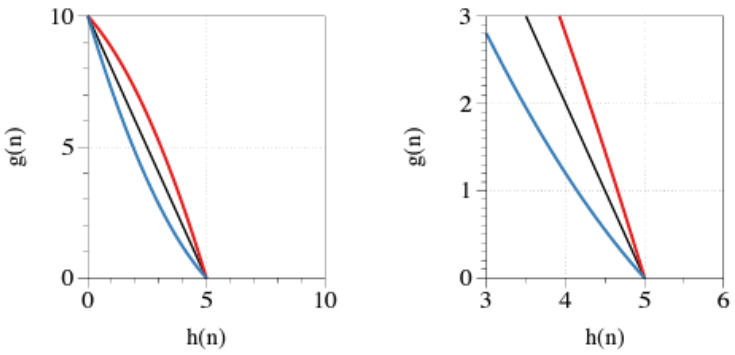
$$f(n) = \Phi(h(n), g(n)) \tag{11}$$

Následně dokázali, že i taková evaluační funkce je suboptimální v rámci dané váhy w. Po vzoru tohoto zápisu poté odvodili nové evaluační funkce, u kterých algoritmus nemusí znovu expandovat již expandované uzly (uzly v seznamu CLOSED), i přesto, že do nich byla nalezena nová a kratší cesta. Tímto se algoritmus používající nové funkce zefektivní. Pro demonstraci výsledků byla nejdříve navrhnuta linární funkce, jež po celou dobu vyhledávání ponechává maximální úroveň suboptimality ve všech krocích konstatní. Tato evaluační funkce koresponduje s klasickým zápisem evaluační funkce algoritmu wA*:

$$\Phi(h, g) = \frac{1}{w}g + h \tag{12}$$

Pomocí této rovnice je snadné demonstrovat, že wA* zůstává w-optimální bez potřeby znovuotevírat již expandované stavy [7].

Dalšími evaluačními funkcemi jsou nelineární evaluační funkce, které umožňují, aby se míra suboptimality v průběhu vyhledávání měnila. Zatímco představená lineární funkce poskytuje konstatní míru suboptimality, Chen a Sturtevant představili nelineární funkci konvexní paraboly, která značně omezuje míru suboptimality na začátku nebo na konci vyhledávání. Tím je zajištěno, že suboptimalita je zaměřena na kritické části vyhledávacího procesu, přičemž na začátku nebo na konci cesty je k dispozici jen malá tolerance. Na následujícím obrázku (viz obr. 14) lze vidět graf takových parabol, které představují tři různé izočáry pro prioritní funkce. Každá z těchto prioritních funkcí vrátí w-suboptimální řešení, ale klade různé požadavky na nalezenou cestu. Vpravo je část křivky zobrazena ve větším detailu. Rovná černá čára uprostřed je wA* s váhou $w = 2$. Protože černá izočára algoritmu wA* je přímá, může v libovolném bodě hledání rozšířit stav, který zvýší g-náklady o 2 a h-náklady sníží o 1, a to při zachování stejné priority. Modrá izočára ve spodní části je zakřivená. U g blížící se nule je její sklon téměř vodorovný. Pokud izočára evaluační funkce vytváří takový tvar, pak prohledávání umožní velmi malou míru suboptimality na začátku každé zkoumané cesty. S rostoucími náklady g však bude povolena větší suboptimalita. Červená čára má opačný průběh. Na začátku hledání je povolena větší míra suboptimality, ale čím více se hledání blíží k cíli, tím přísnější nároky na suboptimalitu jsou.



Obr. 14: Grafy evaluačních funkcí (modrá = XDP, černá = wA*, modrá = XUP, $w = 2$), převzato z [7]

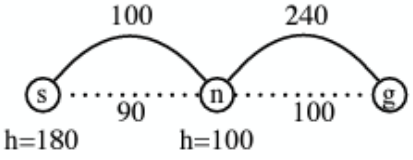
Praktické využití těchto prioritních funkcí je znázorněno na dalším obrázku (viz obr. 15). Předpoklad je zde takový, že wA* prochází od startu s do cíle g s váhou $w = 2$. Z h-nákladů na začátek můžeme usoudit, že horní cesta z s přes n do g (délka cesty 340) se vejde do požadované suboptimální hranice (dvojnásobek hodnoty heuristické funkce, tedy 360). Algoritmus wA* však vyžaduje, aby i část cesty z n do g byla nejvýše dvakrát suboptimální, což v tomto případě není. Evaluační funkce s modrou izočárou na obrázku 14 však připouští menší míru suboptimality v první polovině cesty a větší míru ve druhé polovině, takže by v tomto příkladu byla schopna najít řešení délky 340.

Evaluační funkce znázorněné barevnými izočárami byly odvozeny jako konvexní dolů směřující parabola neboli *Convex Downward Parabola* (XDP) a konvexní nahoru směřující parabola neboli *Convex Upward Parabola* (XUP). Jejich podoba je následující:

$$\Phi_{XDP}(h, g) = \frac{1}{2w}(g + (2w - 1)h + \sqrt{(g + h)^2 + 4wgh}) \tag{13}$$

$$\Phi_{XUP}(h, g) = \frac{1}{2w}(g + h + \sqrt{(g + h)^2 + 4w(w - 1)h^2}) \tag{14}$$

Tyto evaluační funkce v mnoha případech překonávají wA*, a navíc oproti dále představovaným algoritmům nemusí znovuotevírat již expandované uzly.



Obr. 15: Příklad grafu s h-hodnotami uzlů a délkami cest, převzato z [7]

3.3.2 Metody focal

Doposud byly představeny suboptimální metody, které čerpaly informace o dalším směřování v daném kroce z jednoho seznamu, konkrétně seznamu OPEN, ze kterého byl vždy vybrán nejperspektivnější uzel, který se následně expandoval. Dále se však práce bude věnovat kategorii omezených suboptimálních metod, jež ke své funkci využívají další seznam, označovaný jako FOCAL.

Stejně jako v předchozích případech je i zde cílem dosáhnout rovnováhy mezi efektivitou a suboptimalitou. Stěžejní myšlenkou focal metod je aktualizace dvou seznamů najednou. FOCAL množina je ve většině případů považována za perspektivnější z hlediska vedení k cíli a pomocí ní se provádí zkoumání nejslibnějších oblastí prohledávaného prostoru. Zjednodušeně řečeno se focal metody dokáží mnohem snáz soustředit na směřování k cíli, avšak stejně jako u algoritmu wA* za to platí daň ve formě neoptimality. Jelikož se však jedná o omezené algoritmy, stále zde platí, že mezi vstupními parametry je i váha w, která tuto suboptimalitu omezuje dle potřeb.

Algoritmus A* epsilon

Myšlenku FOCAL množiny představili již v roce 1982 J. Pearl a J. H. Kim ve svém článku *Studies in Semi-Admissible Heuristics* [20] a postavili na ni algoritmus A* epsilon. Jeho název se odvíjí od hodnoty epsilon, která definuje váhu w, podle níž se řídí suboptimalita v daném algoritmu:

$$w = 1 + \epsilon, (\epsilon \geq 0) \tag{15}$$

Jak již bylo napsáno, A* epsilon pracuje se dvěma seznamy, tedy OPEN a FOCAL. FOCAL je podseznamem OPEN a obsahuje pouze ty uzly, které se neodchylují od nejlepšího uzlu v OPEN faktorem větším než w:

$$FOCAL = \{n : f(n) \leq (1 + \epsilon) min_{n' \in OPEN} f(n')\} \tag{16}$$

Proces A* epsilon je totožný s algoritmem A* s tím rozdílem, že epsilon verze vybírá uzel ze seznamu FOCAL, a to takový, který má nejnižší heuristickou hodnotou $h(n)$, která odhaduje výpočetní náročnost potřebnou k dokončení hledání.

Důvod pro tento způsob práce je jednoduchý. Všechny uzly v seznamu FOCAL mají přibližně stejné ohodnocení, a proto než trávit čas rozhodováním, který z nich je nejlepší, je smysluplnější věnovat čas výpočtu zbývající části řešení. Je zřejmé, že pro epsilon rovno nule, se A* epsilon redukuje na A*.

Kdyby se z algoritmu odebral seznam OPEN, korespondoval by algoritmus s GBFS algoritmem. To fakticky znamená, že se vždy prostor expanduje velmi rychle pomocí uzlů, které jsou v seznamu FOCAL. Řešení se o dost rychleji blíží výsledku, ale nezaručuje optimalitu. Jakmile se expandace přiblíží limitu pro daný uzel v OPEN seznamu (viz předešlá rovnice), musí být expandovaný i ten a řešení je

svým způsobem dále zkontrolováno klasickým A* algoritmem, což sice zvýší náklady, ale algoritmu dodá jistotu w-optimality [30, 20].

Algoritmus Optimistic search

V roce 2008 se ve své práci *Faster than Weighted A*: An Optimistic Approach to Bounded Suboptimal Search* J. T. Thayer a W. Ruml věnují novému algoritmu, který taktéž vychází z myšlenky dvou seznamů. Algoritmus, který představili, má dvě fáze. První je agresivní část hledání (dle které je algoritmus pojmenovaný), u které není zaručeno, že je w-přípustná (i když obvykle je). Po ní následuje fáze tzv. čištění, v níž se algoritmus snaží prokázat, že řešení získané agresivní částí hledání je přípustné (w-optimální).

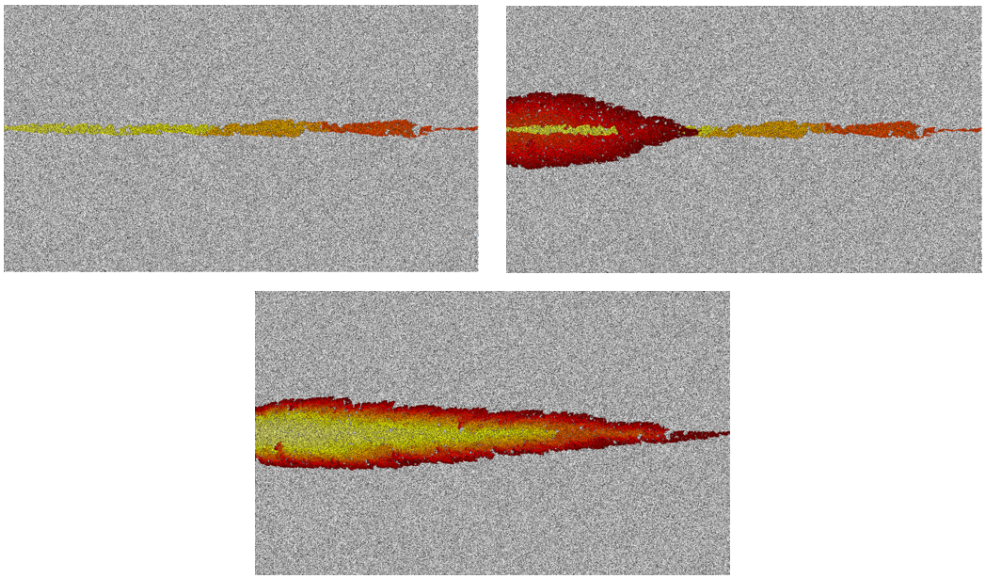
První fáze (optimistická část) je dosud nepředstavený koncept, jelikož algoritmy jako wA* nebo A* epsilon jsou velmi striktní. Žádný z těchto algoritmů nikdy neexpanduje uzel, který nemusí vést k w-optimálnímu řešení. To vede k optimistickému přístupu, kdy je povoleno agresivněji rozšiřovat uzly, aniž bychom měli striktní záruku, že vedou k w-přípustnému řešení. Pro tuto fázi může být zvolena jakákoliv rychlá vyhledávací metoda se seznamem OPEN (například wA* s velkou vahou přesahující vstupní omezení).

Aby se řešení dalo po první optimistické části považovat za w-přípustné, je třeba po nalezení řešení dále expandovat uzly, pomocí algoritmu A*, jenž buď potvrdí w-přípustnost nebo najde lepší řešení. Rozhodně však nepotřebuje hledat celou novou cestu, většinou w-přípustnost potvrdí po pár krocích, a ušetří tak mnoho expandovaných uzlů, které by musel projít v případě použití pouze takového algoritmu [32].

Na následujícím obrázku (viz obr. 16) lze vidět srovnání expandovaných uzlů u první (vlevo nahoře) a druhé (vpravo nahoře) fáze algoritmu Optimistic search. Dole je pro porovnání ten samý problém řešený algoritmem A*. Černá políčka jsou překážky v prostoru, bílá volné uzly k dispozici pro cestu a barevně jsou naznačeny expandované uzly.

Algoritmus 7 Optimistic Search

```
1: function OPTIMISTICSEARCH(start, goal, w)
2:     OPEN ← start
3:     FOCAL ← start
4:     CLOSED ← 0
5:     incumbent ← ∞
6:     while w · f(first on OPEN) ≥ f(incumbent) do
7:         if f'(first on FOCAL) < f'(incumbent) then
8:             n ← remove first from FOCAL
9:             remove first from OPEN
10:        else
11:            n ← remove first from OPEN
12:            remove first from FOCAL
13:        end if
14:        add n to CLOSED
15:        if n is a goal then
16:            incumbent ← n
17:        else
18:            for all child c of n do
19:                if c is duplicated in OPEN then
20:                    if c is better than the duplicate then
21:                        replace copies in OPEN and FOCAL
22:                    end if
23:                else if c is duplicated in CLOSED then
24:                    if c is better than the duplicate then
25:                        add c to OPEN and FOCAL
26:                    end if
27:                else
28:                    add c to OPEN and FOCAL
29:                end if
30:            end for
31:        end if
32:    end while
33: end function
```



Obr. 16: Porovnání algoritmů Optimistic search a A*, převzato z [31]

Algoritmus Improved optimistic search

Základní princip algoritmu Improved optimistic search (IOS) je identický s předchozím algoritmem Optimistic search, ze kterého vychází. Stejně jako předchozí algoritmus používá IOS k hledání a nalezení řešení FOCAL seznam a k prokázání neoptimálnosti nalezeného řešení seznam OPEN. V mnoha případech však lze k důkazu, že řešení je v mezích optimality, použít maximální náklady uzlů z FOCAL seznamu, aniž by bylo nutné rozšiřovat stavy v OPEN seznamu. Algoritmus IOS je tak efektivnější, jelikož znovu neotvírá stavy, které právě Optimistic navrací do seznamu OPEN ze seznamu CLOSED. Navíc má IOS možnost použít i lepší ukončovací podmínku, která často mnohem dříve rozpozná, zda je algoritmus w-optimální (viz pseudokód algoritmu). IOS je také schopen používat širší třídu evaluačních funkcí (XDP, UDP [7]).

Algoritmus Dynamic potential search

Ve své podstatě by poslední představovaný algoritmus mohl mít svoji vlastní kategorii. Jedná se o druh algoritmu, jenž provádí prohledávání na základě potenciálu jednotlivých uzlů. Tento koncept byl představen již v roce 2011 algoritmem Potential search [28]. Tento algoritmus vytvořil svoji speciální podkategorii tzv. suboptimálních metod omezených na základě délky cesty. Ty definují maximální délku cesty C, přičemž její velikost nesmí být při hledání řešení překročena. Tato práce však tuto novou kategorii problémů nepředstavuje, jelikož bylo v roce 2016 ve článku *Dynamic Potential Search – a New Bounded Suboptimal Search Algorithm* od D. Gilona, A. Felnera a R. Sterna [11] prokázáno, že se všechny algoritmy spadající do ní dají jednoduše převést do kategorie klasických omezených suboptimálních algoritmů, do

Algoritmus 8 Improved Optimistic Search

```
1: function IOS(start, goal, w)
2:     OPEN ← start
3:     FOCAL ← 0 [c(I) = ∞]
4:     I ← ∞
5:     while c(I) not w-optimal do
6:         if est. path length of best on FOCAL < c(I) then
7:             expand best from FOCAL
8:             if best == goal then
9:                 I ← path(best)
10:            end if
11:        else
12:            expand best from OPEN
13:            if child c has shorter path to c on FOCAL then
14:                update cost of c on FOCAL
15:            end if
16:        end if
17:    end while
18: end function
```

nichž spadá i Dynamic potential search (DPS) představený tímto článkem. DPS vychází právě z algoritmu Potential search a upravuje ho tak, aby ohraničení suboptimality nebylo provedeno maximální délkou cesty C, nýbrž součinem minimální hodnoty f v OPEN seznamu a požadovaného suboptimálního ohraničení (váhou w). DPS je speciálním případem focal search metod a je tedy i pro tentokrát zaručeno, že vždy nalezne řešení v rámci požadované meze suboptimality.

Slabinou DPS je, že priorita (evaluační funkce) uzlu v OPEN závisí na nejnižší hodnotě v seznamu OPEN, jež se v průběhu hledání zvyšuje. Proto je nutné při každém takovém kroku aktualizovat pořadí v seznamu OPEN, což značně zpomaluje chod algoritmu.

Jak bylo zmíněno, pro výběr prioritního respektive nejlepšího uzlu není použita tzv. evaluační funkce, ale potenciál uzlů, který vychází z následující rovnice:

$$ud(n) = \frac{B \cdot f_{min} - g(n)}{h(n)} \tag{17}$$

kde B je mez suboptimality, f_{min} je uzel s nejmenší hodnotou evaluační funkce v seznamu OPEN. Násobek těchto dvou parametrů definuje mez suboptimality pro daný moment. Čitatel v této rovnici, tedy mez suboptimality pro daný moment od nějž se odečte hodnota $g(n)$, představuje zbývající část suboptimality, kterou po

vydělení $h(n)$ můžeme nazvat jako hodnotu udávající kolik uzlů z celkové cesty již bylo použito.

4 IMPLEMENTACE ALGORITMŮ

K implementaci byly vybrány algoritmy patřící do kategorie omezených suboptimální metod, tedy metod wA* a focal. Tento výběr byl zvolen zejména proto, že narozdíl od neomezených suboptimálních algoritmů lze zaručit míra jejich suboptimality. Respektive se hranice suboptimality nastavuje již jako vstupní parametr, což u neomezených provést nelze. Anytime algoritmy tato práce taktéž nepopisuje a neanalyzuje po praktické stránce, a to zejména proto, že jsou tyto algoritmy na omezených suboptimálních metodách postavené. Tudíž by se jednalo pouze o jiný způsob interpretace výsledků, a to z hlediska poskytnutého časového rámce. Tento přístup by se však nedal blíže porovnávat s omezenými metodami. Mezi vybrané algoritmy byly zvoleny následující:

- A* algoritmus (optimální, pro srovnání výsledků)
- wA* algoritmus (kategorie wA*)
- wA* XDP algoritmus (kategorie wA*)
- wA* XUP algoritmus (kategorie wA*)
- IOS algoritmus (kategorie focal)
- DPS algoritmus (kategorie focal)

4.1 Programovací prostředky

Pro implementaci algoritmů, jejich grafické zobrazení a zpracování výsledků bylo vytvořeno simulační prostředí pomocí programovacího jazyka Python, jenž je od roku 1991 jedním z nejpopulárnějších programovacích jazyků vůbec. Jedná se o open-source projekt, který je k dispozici na všech běžně dostupných platformách [2, 4].

4.1.1 Prostředí aplikace

Základním stavebním kamenem simulačního prostředí jsou dvě hlavní *třídy*. Třída *Node* pro každý jednotlivý uzel grafu. A třída *Planner* pro prostředí řídící celý proces prohledávání. Z toho vyplývá, že jako základní programovací paradigma bylo zvoleno objektově orientované programování (OOP), jež definuje tzv. *objekt*, tedy ucelenou koncepční jednotku, která obsahuje své vnitřní proměnné (*atributy*) a funkce (*metody*). Objekt je instancí třídy, která charakterizuje jeho obecnou strukturu. Každý uzel grafu je tak instancí třídy *Node* s jedinečnou adresou [3].

Tato základní struktura je doplněna o skripty s algoritmy jako takovými a dalšími externími funkcemi. Každý algoritmus je přiložen jako funkce zpracovávající vstupní parametry, mezi něž patří graf ve formě mřížky (definovaný pomocí do sebe vnořených seznamů, tzv. *list*, což je základní datová struktura jazyka Python),

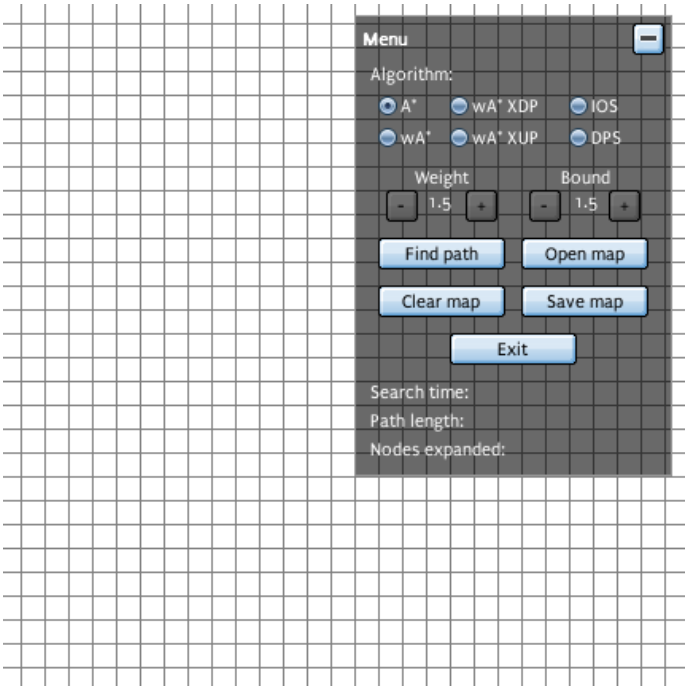
počáteční uzel, cílový uzel a v případě suboptimálních algoritmů vstupuje do funkce také *bound* neboli maximální míra suboptimality (omezení či hranice suboptimality). V případě IOS algoritmu navíc i váha w, jež určuje míru prohledávání prostoru první fáze algoritmu. Vše je doplněno o balíček funkcí s heuristickými metrikami.

K dispozici jsou dále dvě podsložky, */ui* složka pro grafické rozhraní (*user-interface*, viz následující část textu) a */maps*, což je složka sdružující všechny uložené mapy pro snadnější a rychlejší analýzu výsledků a jejich porovnávání.

4.2 Uživatelské rozhraní

Uživatelské prostředí bylo vytvořeno pomocí sady modulů *Pygame*, jež je určená pro vývoj multimediálních aplikací. Poskytuje funkce pro vývoj her či simulačních prostředí. Základními prvky této knihovny jsou kreslení tvarů, načítání a zobrazování obrázků, přehrávání zvuků a hudby, zpracování vstupu z klávesnice a myši a implementace herní logiky prototypů her. Pygame má otevřený zdrojový kód, rozsáhlou dokumentaci, výukové programy a zdroje [26].

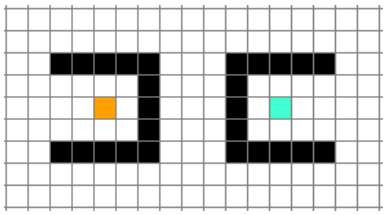
Uživatelské prostředí jako takové je tvořeno mřížkou reprezentující graf, po jehož hranách je prostor prohledávaný. V mřížce se lze pohybovat 8 směry, tedy pro jakýkoliv uzel mimo okrajové platí, že z něj vede cesta do všech 8 sousedních uzlů (2 vertikální cesty, 2 horizontální a 4 diagonální). V pravém horním rohu se nachází menu (viz obr. 17) s možností výběru algoritmu a jeho vstupních parametrů. Dále menu obsahuje tlačítka pro ovládání prostředí a informační sekci.



Obr. 17: Uživatelské rozhraní simulačního prostředí

4.2.1 Ovládání aplikace

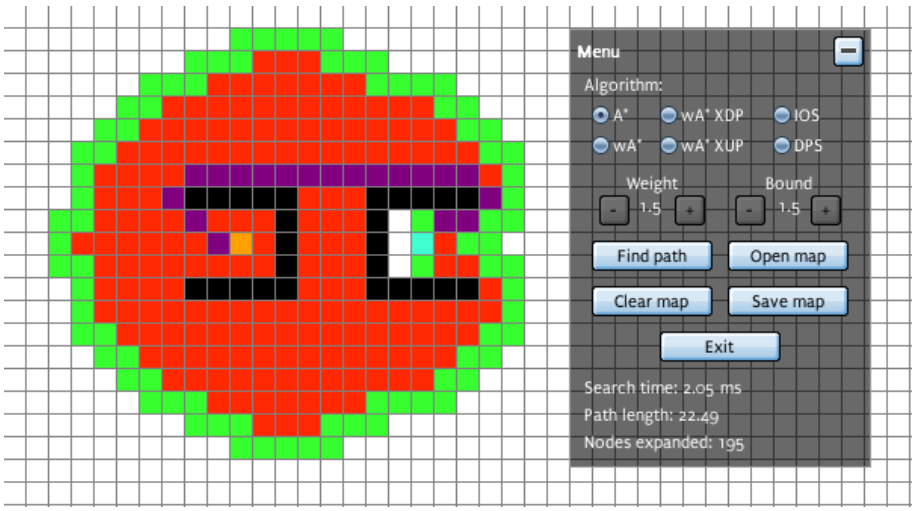
Pomocí levého tlačítka myši se do grafu vkládá startovní uzel (první kliknutí) a cílový uzel (druhé kliknutí). Následně se tím samým tlačítkem do mapy pokládají překážky znázorněné černým pozadím a to buď klikáním nebo tažením myši. Pravým tlačítkem myši se daný uzel vymaže, respektive se uvolní a může jím tak procházet cesta. V aplikaci rovněž fungují tři klávesové zkratky. Pomocí tlačítka „c“ jako (*clear*, vymazat) se mřížka promaže. Pomocí „r“ (*refresh*, obnovit) se odstraní pouze stavy, které byly vytvořeny algoritmem, tedy start, cíl a překážky zůstávají. Klávesa „b“ (*back*, zpět) pak vrací předchozí verzi mřížky, na které byl spuštěn libovolný algoritmus.



Obr. 18: Příklad jednoduché mapy (start = oranžový uzel, cíl = tyrkysový uzel)

V menu se nachází devět tlačítek. Tlačítka plus a minus nastavují velikost váhy w a míry suboptimality *bound* a to v intervalu od 1,5 do 9. Zatímco u A* algoritmu jsou tlačítka deaktivovaná, u ostatních algoritmů se aktivují vždy dle požadovaných vstupních parametrů. Pro spuštění algoritmu, který je vybrán na základě možností v horní části, slouží tlačítko *Find path* (najít cestu). Tlačítko *Clear map* má stejnou funkci jako klávesa „c“, tedy nastavení všech stavů v mřížce jako možné cesty. Simulační prostředí je uzpůsobené i pro ukládání navrhnutých map, k tomu slouží možnost *Save map*. K jejich následnému otevření slouží tlačítko *Open map*. V poslední řadě pak tlačítkem *Exit* uživatel aplikaci vypne.

Ve spodní části menu se nachází informační sekce, která prezentuje výsledky posledního spuštěného algoritmu na dané mapě. Kromě času, za který algoritmus našel cestu, uživatel dostane informaci o délce finální cesty a o počtu expandovaných uzlů. Po nalezení cesty se fialovou barvou zobrazí nejlepší (nejkratší) cesta, kterou daný algoritmus našel (viz obr. 19). Červené uzly jsou expandované uzly, tedy ty v seznamu CLOSED. Zelené uzly jsou uzly ještě neexpandované, ale již objevené, tedy ty v seznamu OPEN. U algoritmu IOS je použita i modrá barva, která značí expandované uzly ve druhé části algoritmu (objeví se pouze, pokud ji algoritmus využije).



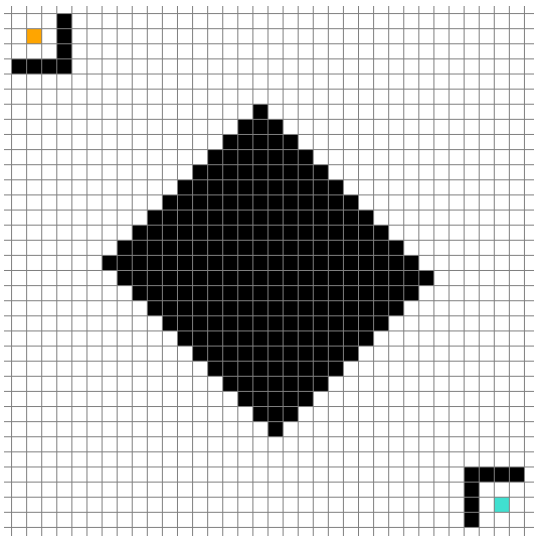
Obr. 19: Grafické provedení nalezení cesty algoritmem A*

5 VÝSLEDKY EXPERIMENTŮ

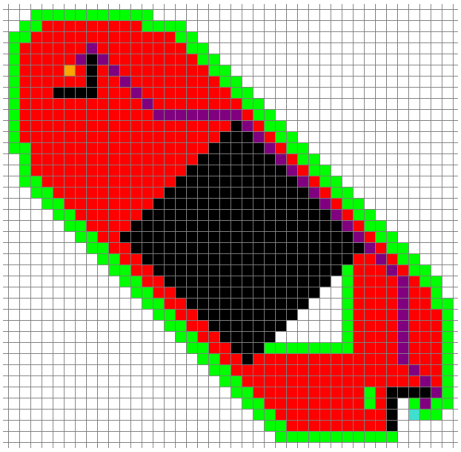
V následující sekci jsou provedeny experimenty na různých mapách a následně jsou vždy porovnány vybrané algoritmy s algoritmem A*. Sledují se tři různé metriky, a to čas, za který algoritmus dosáhl cílového uzlu, dále délka nejkratší nalezené cesty do cíle a v neposlední řadě počet expandovaných uzlů. Poslední zmíněná metrika shrnuje kombinaci časových i prostorových nároků algoritmu při daném nastavení. Co se týče časových nároků, je vždy vybrán nejkratší čas z 20 provedení algoritmu. Cena neboli délka cesty je vypočtena na základě druhu souseda. Pokud se jedná o diagonálního souseda, cesta do něj stojí $\sqrt{2}$. v případě vertikálního či horizontálního sousedství je cena rovna 1.

5.1 Experiment 1

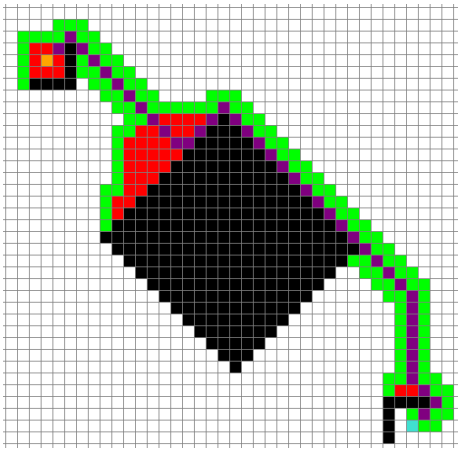
V prvním experimentu je srovnáván algoritmus A* s jeho suboptimální a agresivnější variantou wA*. Mapa byla zvolena tak, aby byl jasně rozpoznatelný rozdíl v principu těchto dvou algoritmů, což potvrzují obrázky 21 a 22. Zatímco A* prohledává všechny možné varianty nejkratší cesty, wA* s vyšší váhou u heuristické funkce mnohem více tíhne ke směru, který je dán polohou cílového uzlu. U wA* algoritmu je váha zároveň mírou suboptimality, a proto čím vyšší je, tím usilovněji směřuje k cíli, ale za cenu kvality výsledné cesty. Zatímco A* efektivně obíhá velkou překážku v cestě, wA* s vyššímu vahami do překážky naráží a až poté ji obíhá. To právě z důvodu, že se cílový uzel nachází za ní a heuristická funkce se tak snaží získat co nejlepší pozici vzhledem k němu.



Obr. 20: Mapa 1



Obr. 21: A* - mapa 1



Obr. 22: wA* (bound = 5), mapa 1

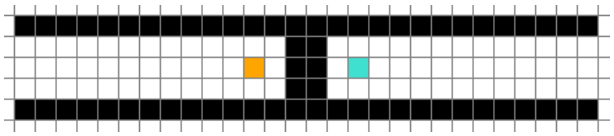
Jak lze vidět v tabulce 3, čím vyšší bound (míra suboptimality a zároveň prohledávací váha) je zvolena, tím menší prostor je potřeba prohledat, jelikož algoritmus rychleji tíhne k cíli. To znamená, že výslednou cestu najde s vyšší vahou rychleji než předchozí verze či algoritmus A*, avšak jak již bylo zmíněno, finální cesta již není optimální. Lze si všimnout, že i přes delší koncovou cestu však ani v jednom z testovaných případů nejde suboptimální výsledek přes přípustnou hodnotu. Navíc při takto jednoduché mapě nepřesáhne přípustnost hranici 1,5 ani při použití váhy 7. Časově se nejrychlejší výsledky algoritmu wA* dostanou na méně než polovinu toho, co potřebuje A*.

Tab. 3: Srovnání algoritmů A* a wA* - mapa 1

A*			**wA***			
délka	**expand.**	**čas [ms]**	**bound**	**délka**	**expand.**	**čas [ms]**
53.60	478	3.07	1.5	53.60	125	1.52
			2	53.60	94	1.47
			3	55.25	89	1.38
			5	56.08	80	1.34
			7	56.08	77	1.33

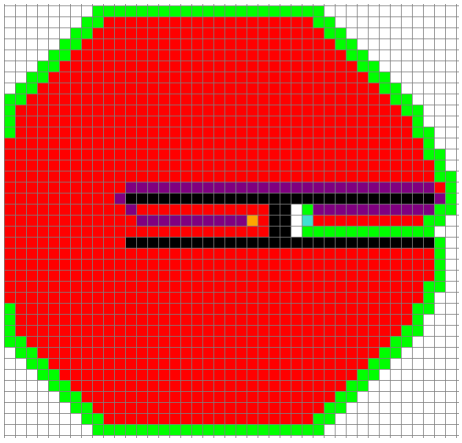
5.2 Experiment 2

Ve druhém experimentu dochází ke srovnání omezených suboptimálních algoritmů typu wA*, tedy wA*, XDP a XUP. I mapa číslo 2 je velmi jednoduchou verzí mřížky a představuje snadnou úlohu pro všechny z nich. Díky tomu však lze rozpoznat základní vlastnosti a rozdíly těchto algoritmů.

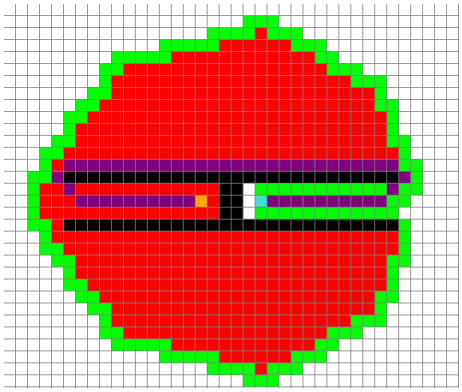


Obr. 23: Mapa 2

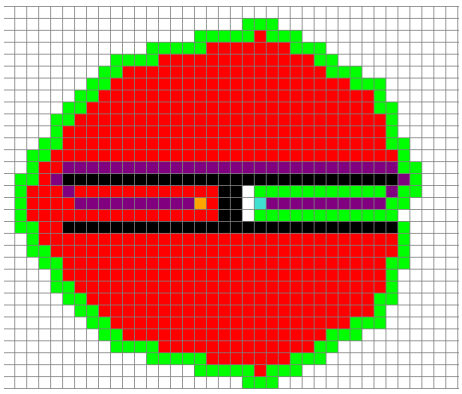
Z následujících čtyřech obrázků je patrné, že všechny algoritmy zvládají najít optimální cestu. To potvrzuje, že i suboptimální algoritmus dokáže najít velmi dobré výsledky. Samozřejmě zde je vše dáno krátkým řešením úlohy. Dále je opět viditelné, že narozdíl od A* algoritmu mají všechny wA* algoritmy větší tendenci tíhnout směrem k cíli. Ten největší rozdíl lze sledovat při opuštění levé části překážky. v sekci o XDP a XUP algoritmu již bylo představena jejich hlavní výhoda, kterou je proměnná míra suboptimality v průběhu prohledávání. XDP algoritmus povoluje pouze malou míru suboptimality na začátku prohledávání, tedy snaží se zezačátku hledat co nejlepší cestu. Jeho nároky na optimalitu se v průběhu uvolňují a na konci jsou naopak nejnižší. V tomto případě to však nepřineslo žádnou větší výhodu. Naopak algoritmus musel zezačátku prohledávat větší část prostoru. Opačně je tomu v případě XUP algoritmu, který zezačátku prohledávání umožňuje zvýšit váhu heuristické funkce a díky tomu nalezne výslednou cestu při mnohem nižších nákladech.



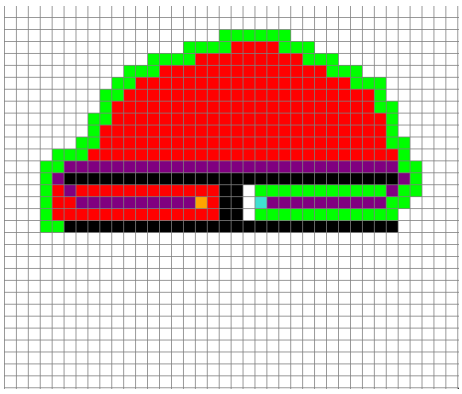
Obr. 24: A* - mapa 2



Obr. 25: wA* (bound = 5) - mapa 2



Obr. 26: XDP (bound = 5) - mapa 2



Obr. 27: XUP (bound = 5) - mapa 2

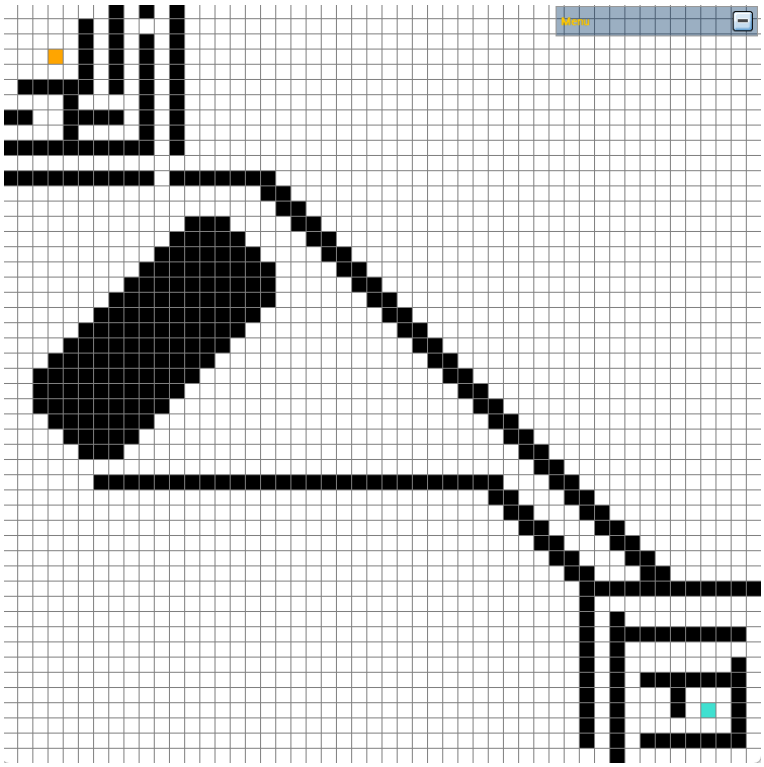
Tabulka 4 potvrzuje toto chování a v případě nižších vah lze pozorovat o něco lepší výsledky u algoritmu XDP. Naopak s rostoucí mírou suboptimality se zlepšují podmínky pro počáteční agresivnější hledání algoritmu XUP, který ve výsledcích naměřených s vyššími vahami naprosto dominuje a stále si drží optimální délku cesty. To vše v trojnásobně kratším čase oproti algoritmu zaručujícímu optimalitu.

Tab. 4: Srovnání algoritmů A*, wA*, XDP, XUP - mapa 2

A*				**wA***		
délka	**expand.**	**čas [ms]**	**bound**	**délka**	**expand.**	**čas [ms]**
55.49	1133	6.05	1.5	**55.49**	929	5.21
			2	**55.49**	807	4.93
			3	**55.49**	652	4.23
			5	**55.49**	529	3.78
			7	**55.49**	270	2.26
XDP				**XUP**		
délka	**expand.**	**čas [ms]**	**bound**	**délka**	**expand.**	**čas [ms]**
55.49	**903**	**5.19**	1.5	**55.49**	937	6.08
55.49	**781**	**4.82**	2	**55.49**	784	5.42
55.49	654	4.75	3	**55.49**	**593**	**4.15**
55.49	552	4.06	5	**55.49**	**268**	**2.35**
55.49	511	3.82	7	**55.49**	**257**	**2.09**

5.3 Experiment 3

Ve třetím experimentu je využita již o něco komplexnější mapa mřížky. Srovnávány jsou omezené suboptimální algoritmy typu focal, konkrétně IOS a DPS algoritmus. I tentokrát je vše vloženo do kontextu díky optimálnímu algoritmu A*.



Obr. 28: Mapa 3

K nalezení cesty v mapě 3 je u všech algoritmů vysledována značná prostorová náročnost. Algoritmus A* potřebuje k nalezení optimální cesty téměř celou velikost mřížky, jejíž plocha čítá 40 x 40 uzlů, a musí tak expandovat 2184 z nich. To se samozřejmě projevuje na rychlosti nalezení optimální cesty. Na obrázku 30 se nachází výsledný stav algoritmu DPS, který rozhodně nevyužívá tolik prostoru k nalezení řešení. S mírou suboptimality $bound = 5$ se však projevuje o dost vyšší cena za finální cestu a taktéž z časových nároků tohoto algoritmu jde vyhodnotit, že se projevuje opakované přerovnávání uzlů v seznamu OPEN, které je prováděno v každém cyklu. Výhodou tohoto algoritmu tak zůstává pouze nižší prostorová náročnost.

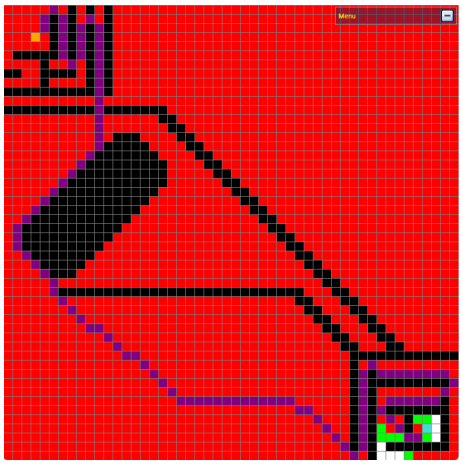
Obrázky 31 a 32 zachycují konečnou podobu algoritmu IOS, avšak každý v jiném nastavení výchozích parametrů. Jak zde, tak v následujících experimentech, platí, že se vyhledávací váha w (*weight*) řídí doporučeným vztahem $w = 2 \cdot bound - 1$. V prvním obrázku je míra suboptimality zvolena jako $bound = 5$. Toto nastavení algoritmu nevyžaduje spuštění kontrolní a opravné druhé fáze algoritmu, která by kontrolovala, zda je algoritmus opravdu w-přípustný (v tomto případě bound-

přípustný). Naopak v druhém případě nastavení $bound = 2$ již algoritmus využívá z části svoji druhou fázi. Ta je znázorněna modrou barvou. Tyto uzly jsou tedy znovuexpandovány, a algoritmus tak pro zajištění dané míry suboptimality platí časem i prostorem.

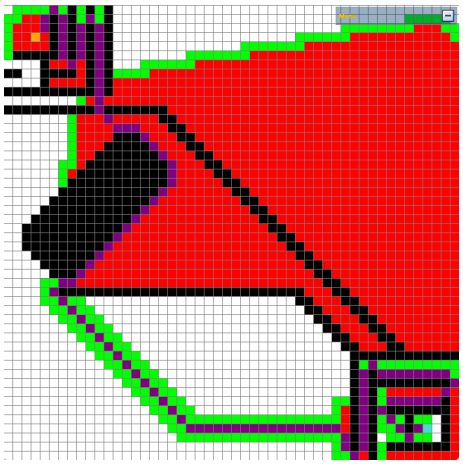
Z tabulky 5 je patrné, že v tomto konkrétním případě by se oproti A* algoritmu časově vyplatila pouze jedna verze IOS algoritmu. S rostoucí mírou suboptimality však klesají nároky na prohledané uzly a zároveň výsledná cena cesty se nijak zásadně neliší od optimální.

Tab. 5: Srovnání algoritmů A*, IOS a DPS - mapa 3

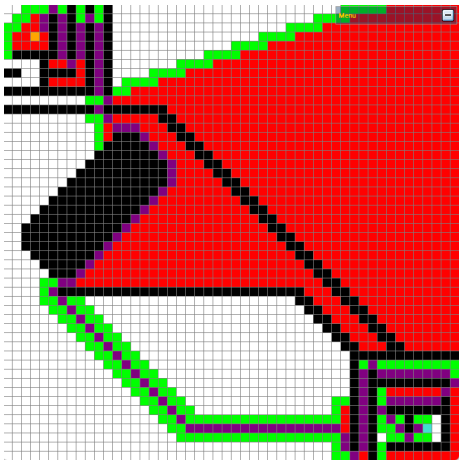
A*						
délka	**expand.**	**čas [ms]**				
129.30	2184	10.41				
IOS				**DPS**		
délka	**expand.**	**čas [ms]**	**bound**	**délka**	**expand.**	**čas [ms]**
134.37	**2125**	**18.67**	1.5	**129.30**	2139	119.49
134.37	**1261**	**10.64**	2	**129.88**	1863	105.39
135.20	**1137**	**10.14**	3	**134.37**	1232	69.37
135.78	1170	**10.53**	5	**134.95**	**1159**	66.08
135.78	1170	**10.53**	7	**134.95**	**1134**	64.21



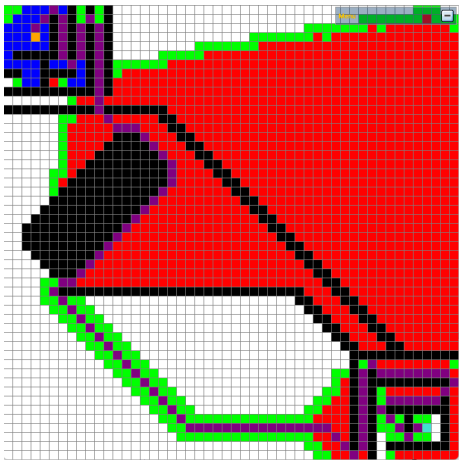
Obr. 29: A* - mapa 3



Obr. 30: DPS (bound = 5) - mapa 3



Obr. 31: IOS (b = 5, w = 9) - mapa 3



Obr. 32: IOS (b = 2, w = 3) - mapa 3

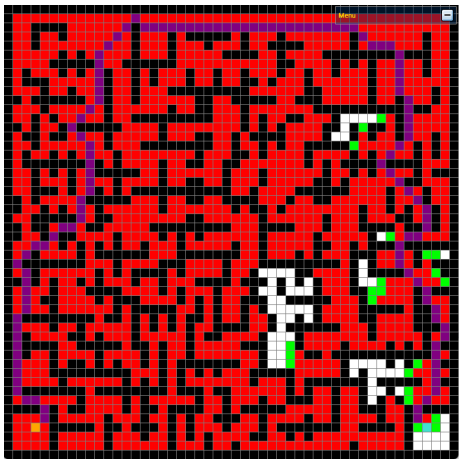
5.4 Experiment 4

V pořadí posledním experimentu byly srovnány všechny algoritmy dohromady a to na dvou různých mapách, jež lze označit za komplexní. Jedná se o dvě „bludiště“, přičemž mapa 4 je svým charakterem spíše uzavřená a obsahuje menší množství možností průchodu směrem k cíli. Mapa 5 je otevřená a nabízí algoritmům mnohem širší spektrum cest.

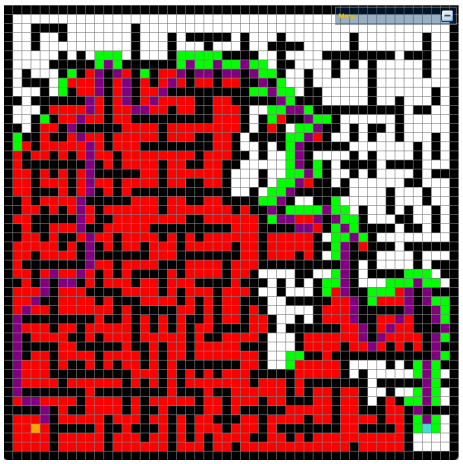
Obr. 33: Mapa 4

Zatímco algoritmus A* si musí ověřit svoji optimalitu expandováním téměř celého prostoru, u vyšších vah suboptimálních algoritmů tomu tak není. Metody wA* i focal využívají své váhy u heuristické funkce a kromě algoritmu DPS, který přerovnává pořadí OPEN seznamu, nalézají výslednou cestu rychleji než A*. V takto komplexním prostoru již lze pozorovat odlišné postupy jednotlivých algoritmů. XDP a XUP nebyly vyobrazeny, jelikož se jejich průběh v případě této mapy o mnoho nelišil od wA* algoritmu.

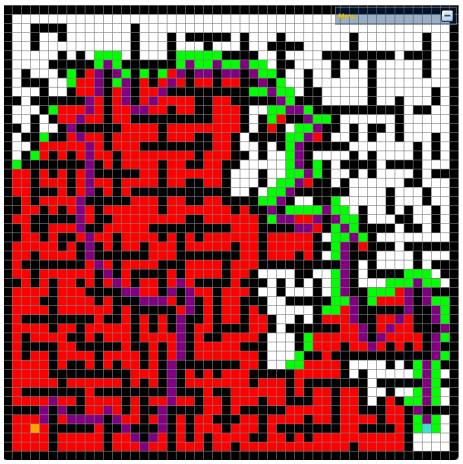
Výsledky v tabulce 6 poukazují na to, že v komplexních mapách je pro různé vstupní parametry výhodné použít různé algoritmy. Co se týče prostorových nároků, exceluje mezi algoritmy nejpomalejší z nich, tedy DPS algoritmus. IOS naopak prohledává prostor nejefektivněji při vyšších vahách, kdy nemusí potvrzovat svoji přípustnost pomocí druhé fáze prohledávání. Mezi nejrychlejší algoritmy se však řadí wA* varianty, které v kombinaci poměru paměťových nároků a času dosahují nejlepších výsledků. Časové rozdíly zde však nejsou příliš markantní, jelikož charakter mapy 4 je uzavřený a nemusí se tak prohledávat mnoho variant.



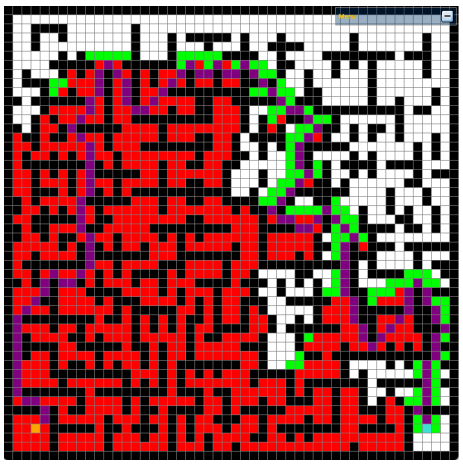
Obr. 34: A* - mapa 4



Obr. 35: wA* (bound = 5) - mapa 4



Obr. 36: IOS (b = 5, w = 9) - mapa 4

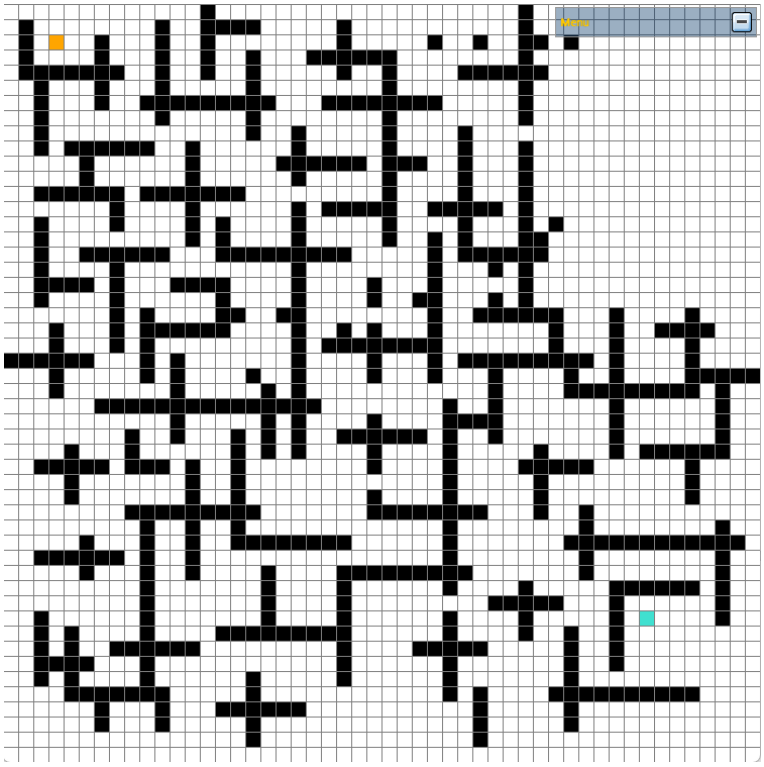


Obr. 37: DPS (bound = 5) - mapa 4

Tab. 6: Srovnání všech algoritmů - mapa 4

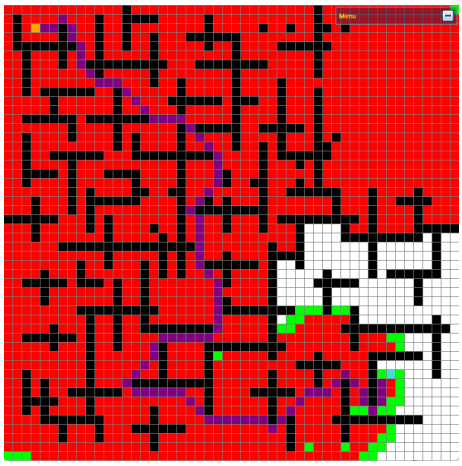
A*				**wA***		
délka	**expand.**	**čas [ms]**	**bound**	**délka**	**expand.**	**čas [ms]**
137.23	1407	7.12	1.5	143.37	1126	5.39
			2	143.37	926	5.06
			3	148.92	869	**4.47**
			5	**149.75**	847	**4.41**
			7	**148.34**	843	**4.29**
XDP				**XUP**		
délka	**expand.**	**čas [ms]**	**bound**	**délka**	**expand.**	**čas [ms]**
143.37	**991**	**5.32**	1.5	140.78	1279	6.84
144.20	**896**	**4.95**	2	144.78	971	5.14
148.10	870	4.98	3	148.10	**865**	4.63
149.75	849	4.77	5	172.72	844	4.79
149.75	846	4.72	7	172.72	836	4.60
IOS				**DPS**		
délka	**expand.**	**čas [ms]**	**bound**	**délka**	**expand.**	**čas [ms]**
147.51	1458	10.62	1.5	**137.23**	1451	81.68
148.92	1161	9.11	2	**137.23**	1451	81.67
172.72	938	7.34	3	**139.81**	1451	85.33
172.72	**827**	6.6	5	**149.75**	855	50.15
172.72	**827**	6.6	7	**148.34**	844	47.94

Posledním procházeným prostorem je mapa 5. Tato otevřenější varianta nabízí algoritmům větší svobodu v rozhodování o jejich dalším směřování. I když se jedná o vcelku nenáročný a malý prostor, rozdíly v jednotlivých algoritmech jsou vcelku markantní.

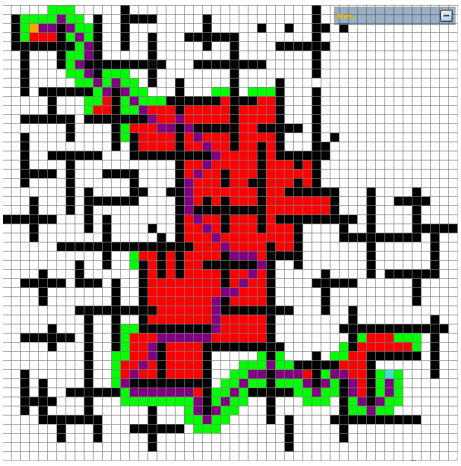


Obr. 38: Mapa 5

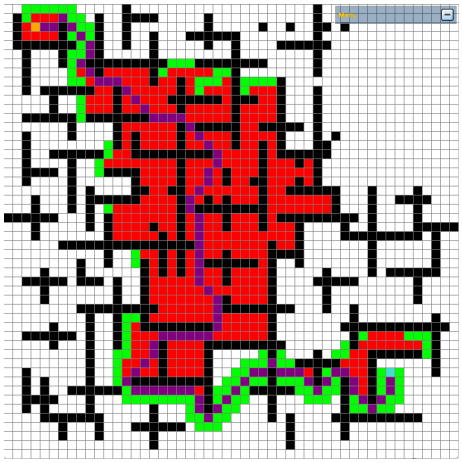
Nejlepší výsledky v jednotlivých kategoriích si algoritmy mezi sebou rozdělily v závislosti na vstupních parametrech. I v posledním případě se všechny algoritmy kromě DPS vměstnaly časově i prostorově pod čísla algoritmu A*. Rychlost, respektive čas dokončení, je u těchto algoritmů velmi podobná a rozdíly jsou jen v rámci několika jednotek milisekund. U velikosti prohledaného prostoru už jsou s vyšší mírou suboptimality rozdíly mnohem větší. Nejlepší výsledky z tohoto pohledu vykazuje algoritmus IOS. Zajímavostí je, že i při nejvyšších vahách se ani u jednoho algoritmu výsledek nedostal přes chybu větší než 1,15násobek optimální cesty.



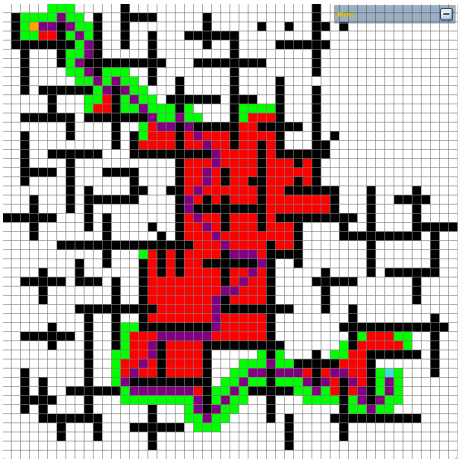
Obr. 39: A* - mapa 5



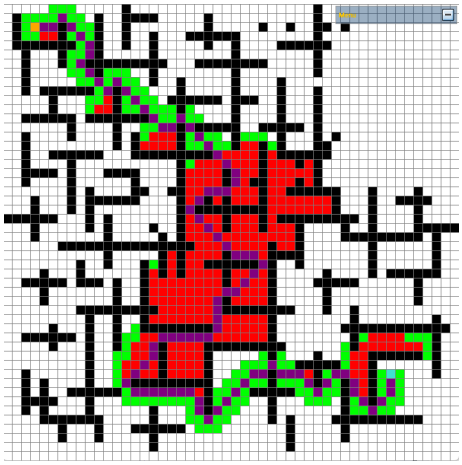
Obr. 40: wA* (bound = 5) - mapa 5



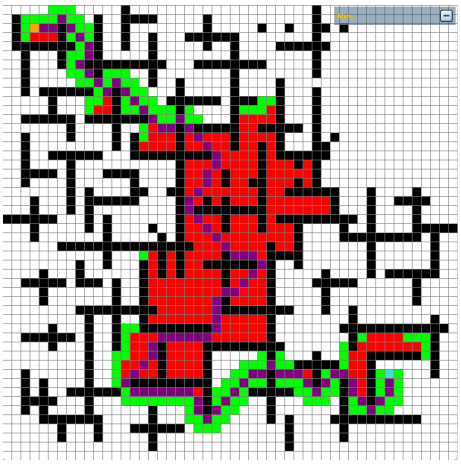
Obr. 41: XDP (b = 5) - mapa 5



Obr. 42: XUP (b = 5) - mapa 5



Obr. 43: IOS (b = 5, w = 9) - mapa 5



Obr. 44: DPS (bound = 5) - mapa 5

Tab. 7: Srovnání všech algoritmů - mapa 5

A*				**wA***		
délka	**expand.**	**čas [ms]**	**bound**	**délka**	**expand.**	**čas [ms]**
101.57	1686	8.19	1.5	103.23	1289	**6.66**
			2	104.05	1011	5.63
			3	104.88	828	5.10
			5	112.20	381	2.81
			7	112.20	356	2.57
XDP				**XUP**		
délka	**expand.**	**čas [ms]**	**bound**	**délka**	**expand.**	**čas [ms]**
103.23	**1253**	7.24	1.5	104.05	1442	7.88
103.23	1095	6.59	2	104.88	922	5.42
104.64	957	5.91	3	105.71	485	**3.78**
104.64	484	3.73	5	112.20	355	**2.63**
110.54	437	3.26	7	115.02	340	**2.54**
IOS				**DPS**		
délka	**expand.**	**čas [ms]**	**bound**	**délka**	**expand.**	**čas [ms]**
103.23	1525	12.58	1.5	**101.57**	1896	104.94
110.54	**417**	**4.26**	2	104.05	1403	78.84
115.02	**349**	3.82	3	106.30	809	45.89
115.02	**331**	3.58	5	112.20	357	21.02
115.02	**331**	3.58	7	115.02	345	20.25

6 ZÁVĚR

Cílem této diplomové práce bylo provedení analýzy přístupů k plánování cesty pomocí klasických a suboptimálních metod prohledávání a implementace vybraných suboptimálních algoritmů. V neposlední řadě bylo třeba provést a vyhodnotit ověřovací a srovnávací experimenty.

V teoretické části práce představila základní metody pro plánování cesty, kde byly nejprve popsány klasické metody. A v další části se diplomová práce zabývala detailnějším popisem a kategorizací různých suboptimálních metod, přičemž bylo poukázáno na vlastnosti těchto algoritmů, které různým způsobem zefektivňují průchod prostorem a hledání cesty jako takové.

V rámci implementační části se práce zabývala algoritmy A*, weighted A* (wA*), weighted A* XDP, respektive XUP, s evaluační funkcí ve tvaru konvexní paraboly směřující dolů, respektive nahoru. Dále také dvěma algoritmy z kategorie focal, tedy algoritmy vylepšeného optimistického hledání (IOS) a dynamického vyhledávání pomocí potenciálu jednotlivých uzlů (DPS).

V experimentální části byly provedeny čtyři experimenty ověřující vlastnosti a efektivitu jednotlivých algoritmů. V každém experimentu byly dané algoritmy srovnány a vyhodnoceny dle svých předpokladů. To vše bylo provedeno v simulačním prostředí vytvořeném v jazyce Python.

V prvním a druhém experimentu byly s optimálním algoritmem A* srovnány omezené suboptimální algoritmy kategorie wA*, které oproti jiným prohledávacím metodám konvergují k cíli mnohem rychleji a díky tomu jsou schopny nalézt cesty velmi blízké optimální cestě v mnohem kratším čase. V každém experimentu proběhly testy při několika nastaveních vstupních parametrů.

Ve třetím experimentu byly testovány omezené suboptimální metody kategorie focal, jež ve svojí konstrukci používají více seznamů prioritizujících pořadí prozkoumávaných uzlů grafu, které taktéž urychlují a hlavně zlevňují průchod prostorem.

Čtvrtý experiment obsahující dva typy komplexních prostorů srovnal všechny implementované algoritmy a dokázal jejich univerzální použití. Při ztrátě optimality finální cesty suboptimální algoritmy poskytují mnohem efektivnější průchod prostorem z hlediska časových i paměťových nároků.

SEZNAM POUŽITÉ LITERATURY

[1] *Path Planning* [online]. [cit. 2023-05-17]. Dostupné z: https://fab.cba.mit.edu/classes/865.21/topics/path_planning/.

[2] *Python.org* [online]. Dostupné z: https://www.python.org/.

[3] *Object-oriented programming* [online]. Květen 2023. Page Version ID: 1156935277. Dostupné z: https://en.wikipedia.org/w/index.php?title=Object-oriented_programming&oldid=1156935277.

[4] *Python* [online]. Květen 2023. Page Version ID: 22815118. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Python&oldid=22815118.

[5] Abd Algfoor, Z., Sunar, M. S. a Kolivand, H. A Comprehensive Study on Pathfinding Techniques for Robotics and Video Games. *International Journal of Computer Games Technology*. duben 2015, sv. 2015, s. 736138. DOI: 10.1155/2015/736138. ISSN 1687-7047. Publisher: Hindawi Publishing Corporation. Dostupné z: https://doi.org/10.1155/2015/736138.

[6] Arora, S. a Barak, B. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[7] Chen, J. a Sturtevant, N. R. Conditions for Avoiding Node Re-expansions in Bounded Suboptimal Search. *International Joint Conference on Artificial Intelligence (IJCAI)*. 2019. Dostupné z: https://webdocs.cs.ualberta.ca/~nathanst/papers/chen2019conditions.pdf.

[8] Chen, J., Sturtevant, N. R., Doyle, W. a Ruml, W. Revisiting Suboptimal Search. *Symposium on Combinatorial Search (SoCS)*. 2019, s. 18–25. Dostupné z: https://webdocs.cs.ualberta.ca/~nathanst/papers/chen2019revisiting.pdf.

[9] Edelkamp, S. a Schrödl, S. *Heuristic Search - Theory and Applications*. Leden 2012. ISBN 978-0-12-372512-7.

[10] Foudil, C., DJEDI, N., Sanza, C. a Duthen, Y. Path Finding and Collision Avoidance in Crowd Simulation. *CIT*. Leden 2009, sv. 17, s. 217–228. DOI: 10.2498/cit.1000873.

[11] Gilon, D., Felner, A. a Stern, R. Dynamic potential search - a new bounded suboptimal search. In: *Ninth Annual Symposium on Combinatorial Search*. 2016.

[12] Gini, M. *Important issues about Search Algorithms* [online]. 2010 [cit. 2023-05-12]. Dostupné z: https://www-users.cse.umn.edu/~gini/4511/search.

[13] Hansen, E. A., Zilberstein, S. a Danilchenko, V. A. *Anytime Heuristic Search: First Results*. 97-50. Computer Science Department, University of Massachussetts Amherst, 1997. Dostupné z: http://rbr.cs.umass.edu/shlomo/papers/HZDtr9750.html.

[14] Jones, M. *Artificial Intelligence: A Systems Approach*. Jones & Bartlett Learning, 2008. ISBN 9780763773373.

[15] Likhachev, M., Gordon, G. J. a Thrun, S. ARA\ast : Anytime A\ast with Provable Bounds on Sub-Optimality. In: Thrun, S., Saul, L. a Schölkopf, B., ed. *Advances in Neural Information Processing Systems*. MIT Press, 2003, sv. 16. Dostupné z: https://proceedings.neurips.cc/paper_files/paper/2003/file/ee8fe9093fbbb687bef15a38facc44d2-Paper.pdf.

[16] Maňáková, L. *Pokročilé metody plánování cesty mobilního robotu*. 2020. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství. Dostupné z: https://www.vut.cz/studenti/zav-prace?zp_id=125057.

[17] Patel, A. *Heuristics* [online]. [cit. 2023-05-15]. Dostupné z: http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html.

[18] Patel, A. *Map representations* [online]. [cit. 2023-05-15]. Dostupné z: http://theory.stanford.edu/~amitp/GameProgramming/MapRepresentations.html.

[19] Pearl, J. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. USA: Addison-Wesley Longman Publishing Co., Inc., 1984. ISBN 0201055945.

[20] Pearl, J. a Kim, J. H. Studies in semi-admissible heuristics. *IEEE transactions on pattern analysis and machine intelligence*. IEEE. 1982, č. 4, s. 392–399.

[21] Pedamkar, P. *Uninformed Search | Various types of Uninformed Search Algorithms* [online]. Březen 2020 [cit. 2023-05-09]. Dostupné z: https://www.educba.com/uninformed-search/.

[22] Pohl, I. Heuristic search viewed as path finding in a graph. *Artificial intelligence*. Elsevier. 1970, sv. 1, 3-4, s. 193–204.

[23] Richter, S., Thayer, J. a Ruml, W. The Joy of Forgetting: Faster Anytime Search via Restarting. *Proceedings of the International Conference on Automated Planning and Scheduling*. květen 2021, sv. 20, č. 1, s. 137–144.

DOI: 10.1609/icaps.v20i1.13412. Dostupné z: https://ojs.aaai.org/index.php/ICAPS/article/view/13412.

[24] Ruml, W. *Suboptimal Heuristic Search*. 2022. International Symposium on Combinatorial Search (SoCS-22). Dostupné z: https://www.cs.unh.edu/~ruml/papers/index.html.

[25] Russell, S. a Norvig, P. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2010. ISBN 978-0136042594.

[26] Shinners, P. *Pygame* [online]. [cit. 2023-05-23]. Dostupné z: https://www.pygame.org/.

[27] Steiblin, C. *Bounded Suboptimal Search for Classical Planning*. 2020. Basilej: University of Basel, Department of Mathematics and Computer Science. Dostupné z: https://ai.dmi.unibas.ch/theses.html.

[28] Stern, R., Puzis, R. a Felner, A. Potential Search: A Bounded-Cost Search Algorithm. In:. Leden 2011.

[29] Sturtevant, N. R. *Moving AI Lab* [online]. [cit. 2023-05-16]. Dostupné z: https://www.movingai.com/index.html.

[30] Sturtevant, N. R. *Single-Agent Search - Suboptimal Search Algorithms* [online]. [cit. 2023-05-12]. Dostupné z: https://movingai.com/SAS/SUB/.

[31] Thayer, J. T. a Ruml, W. *A Survey of Suboptimal Search Algorithms*. 2011. International Conference on Automated Planning and Scheduling (ICAPS-11). Dostupné z: https://www.cs.unh.edu/~ruml/papers/index.html.

[32] Thayer, J. T. a Ruml, W. Faster than Weighted A*: An Optimistic Approach to Bounded Suboptimal Search. In: *ICAPS*. 2008, s. 355–362.

[33] Wetten, G.-A. *Combining Novelty-Guided and Bounded Suboptimal Search*. 2017. Basilej: University of Basel, Department of Mathematics and Computer Science. Dostupné z: https://ai.dmi.unibas.ch/theses.html.

SEZNAM OBRÁZKŮ

SEZNAM TABULEK