Obsah

1. Prohledávání stavového prostoru

1.1.Základní informace

Následující text je součástí učebních textů předmětu Umělá inteligence a je určen hlavně pro studenty Matematické biologie. Kapitola shrnuje třídu metod umělé inteligence nazývané souhrnně metodami pro prohledávání stavového prostoru. Jsou popsány základní neinformované metody, i metody založené na heuristických algoritmech, včetně jejich vzájemného porovnání.

1.2.Výstupy z učení

Zvládnutí učebního textu umožní studentům:

- Porozumět a definovat základní pojmy z oblasti prohledávání stavového prostoru
- Algoritmicky popsat jednotlivé prohledávací metody
- Provést vzájemné porovnání a zhodnocení prohledávacích algoritmů s ohledem na jejich úplnnost, prostorovou a časovou složitost a optimálnost
- Řešit modelové úlohy pomocí aplikace prohledávacích algoritmů

1.3.Úvod

Tato kapitola se zabývá množinou algoritmů umělé inteligence, které shrnujeme pod společné pojmenování jako algoritmy pro prohledávání stavového prostoru. Nejprve je třeba objasnit, které třída úloh je za pomoci těchto algoritmů obvykle řešena. Jedná se zpravidla o řešení problémů, kde na základě dostupných informací nejsme schopni dané řešení nalézt přímým výpočtem, nebo je realizace výpočtu pro nalezení řešení neúměrně obtížná.

Příkladem může být nalezení nejkratší cesty mezi dvěma městy, výhra v šachové partii, nebo vyřešení hlavolamů „Hanoiské věže“ či „Loydova patnáctka“. Neexistence snadného řešení realizovaného přímým výpočtem ještě neznamená, že nejsme schopni definovat řídící strategii, která dané řešení nalezne. Například v případě hlavolamu „Loydova patnáctka“ se snažíme z počátečního nastavení 15 kamenů v mřížce 4x4 seřadit tyto očíslované kameny podle velikosti a dosáhnout tak řešení hlavolamu. Naše strategie může být i taková, že se budeme pokoušet náhodně realizovat metodou pokus omyl tahy, které jsou v danou chvíli přípustné. Postupně tak z počáteční konfigurace dovolenými tahy generujeme konfigurace další a vždy hodnotíme, zda je daná konfigurace řešením (kameny jsou seřazeny) či nikoli. Zvolili jsme tedy řídící strategii, která vždy náhodně zvolí jeden z možných tahů a přesune kámen. Takový postup je sice možný, ale není příliš efektivní. Algoritmy pro prohledávání stavového prostoru se snaží tyto prohledávací strategie optimalizovat.

1.4.Definice stavového prostoru

Stavovým prostorem S_p můžeme rozumět dvojici $S_p = (S,\Phi)$, kde $S = \{s\}$ je množina všech možných stavů, ve kterých se může úloha nacházet a $\Phi = \{ \varphi \}$ je množina operátorů pro přechod mezi jednotlivými stavy. Platí, že k-tého stavu s_k dosáhneme aplikací operátoru přechodu φ_{ik} na stav s_i, s_i je tedy předchůdcem stavu s_k a s_k je naopak následníkem stavu s_i.

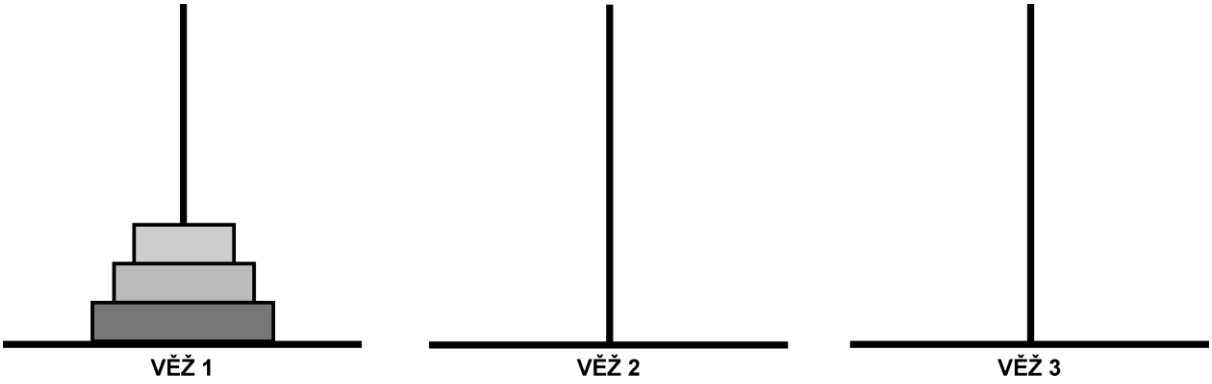
$s_k = \varphi_{ki}(s_i)$

Počáteční stav úlohy, ve kterém se nachází při zahájením hledání řešení, označujeme jako s_0.

Dále definujeme množinu G = {g} cílových stavů, které představují hledané řešení. To může být jedno jediné, cílových stavů představujících řešení však může být i více. Množina G je podmnožinou množiny S.

1.1.1. Reprezentace stavového prostoru

Definici stavového prostoru a možnosti jeho kódování si objasníme na příkladu řešení hlavolamu „Hanoiské věže".
Princip úlohy spočívá v tom, že máme k dispozici tři tyčky a na nich navlečeny tři disky různých průměrů.



***Obr. 1** Hlavolam „Hanoiské věže".*

Úkolem řešitele je přemístit všechny kotouče z první věže na druhou a to za pomoci dočasného odkládání kotoučů na třetí věž. Pro uspořádání a přesuny disků platí následující pravidla:

- V jednom tahu lze přemístit jen jeden kotouč
- Jeden tah je složen z uchopení kotouče z vrcholu jedné věže a z jejího přemístění na vrchol věže jiné
- Je zakázáno položit větší kotouč na menší

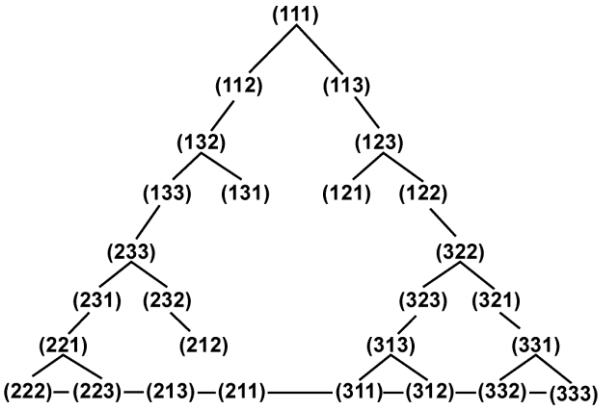
Počátečním stavem s_0 je tedy konfigurace, kdy máme všechny kotouče na první věži seřazené od největšího po nejmenší. Množina cílových stavů zde obsahuje právě jeden stav g a to stav, kdy jsou všechny kotouče korektně uspořádány na druhé věži.

Zaveďme si kódování, které bude vyjadřovat konkrétní stav řešení úlohy pomocí uspořádané trojice čísel, kde pozice čísla v závorce určuje disky od největšího k nejmenšímu a hodnota čísla na dané pozici vyjadřuje, na které věži se daný disk nachází. Počáteční stav s_0 tedy můžeme zapsat jako (111), pokud přesuneme nejmenší kotouč na druhou věž, bude stav kódován jako (112), cílový stav g pak jako (2,2,2).

Celkově se úloha může nacházet v 3^n stavů, kde n představuje počet disků, v našem případě obsahuje tedy množina všech možných stavů S = {s} 27 prvků.

Množina operátorů má v našem případě 6 prvků „přesuň vrchní disk z věže k na věž i", tedy $\Phi = \{ \varphi_{12}, \varphi_{13}, \varphi_{21}, \varphi_{23}, \varphi_{31}, \varphi_{32} \}$

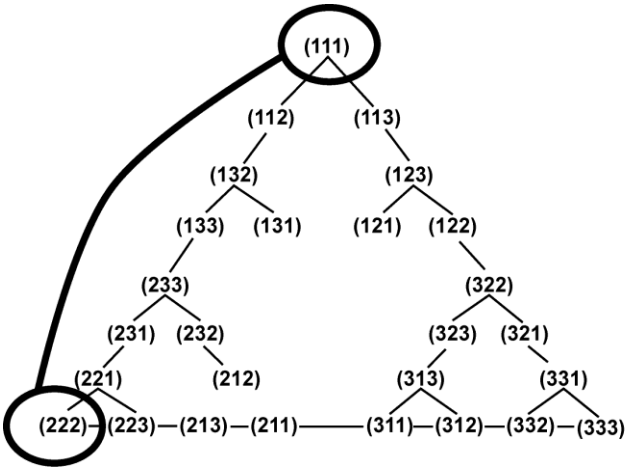
Jednotlivé stavy je možné uspořádat do podoby orientovaného grafu, kdy kořenovým uzlem je počáteční stav a z něj jsou pak aplikací všech přípustných operátorů generovány postupně stavy následující. Přípustný operátor je takový, jehož aplikaci nezakazují pravidla úlohy, například „Je zakázáno položit větší kotouč na menší“.



Obr. 2 *Stavový prostor hlavolamu „Hanoiské věže“.*

1.1.2. Úloha ve stavovém prostoru

V případě jednoduché úlohy „Hanoiských věží“je možné bez problému generovat celý její stavový prostor a zobrazit jej v podobě orientovaného grafu. Řešením úlohy ve stavovém prostoru tedy rozumíme nalezení takové posloupnosti operátorů, kdy nás jejich postupná aplikace přivede z počátečního stavu s_0 do některého z cílových stavů množiny G. V našem případě tedy cestu ze stavu (111) do stavu (222).



Obr. 3 *Stavový prostor hlavolamu „Hanoiské věže“, cílový stav.*

Počet aplikací operátorů nazýváme počtem kroků či lépe délkou cesty. Nalezenou posloupnost operátorů, tedy vlastní řešení pak nazýváme nalezenou cestou z počátečního do cílového stavu. Cílových stavů i nalezených cest k jejich řešení může být více a proto se zpravidla snažíme nalézt cestu z nějakého pohledu optimální – cestu s minimální cenou. Cena aplikace jednotlivých operátorů množiny Φ tedy nemusí být pro všechny tyto operátory vždy shodná.

Reálnou úlohu ve stavovém prostoru tedy řešíme následně:

- Zavedeme si kódování jednotlivých stavů.

- Identifikujeme operátory a omezující podmínky pro jejich aplikaci.
- Identifikujeme cílové stavy představující řešení úlohy.
- Zvolíme počáteční stav. Zpravidla je dán vlastním zadáním úlohy.
- Zvolíme strategii procházení jednotlivých stavů, tedy strategii aplikace jednotlivých operátorů – prohledávání prostoru.
- Aplikujeme zvolenou strategii, začínáme v počátečním stavu.
- V každém kroku kontrolujeme, zda právě dosažený stav (respektive cesta k němu) není hledaným řešením.

Stavový prostor může být velmi rozsáhlý či obecně nekonečný a není tedy vždy možné v konečném čase prověřit všechny možné stavy a přechody mezi nimi, jak tomu bylo v případě předchozí úlohy. Zde tedy nastupují různé strategie pro prohledávání stavového prostoru, které se snaží tento proces optimalizovat. Tyto metody si přiblížíme v následujících odstavcích.

1.5.Metody prohledávání

V následující kapitole si přiblížíme stručně základní metody prohledávání stavového prostoru, které se vyskytují v řadě modifikací. Obecně můžeme metody pro prohledávání stavového prostoru rozdělit následně:

- Neinformované
- Informované
- Lokální

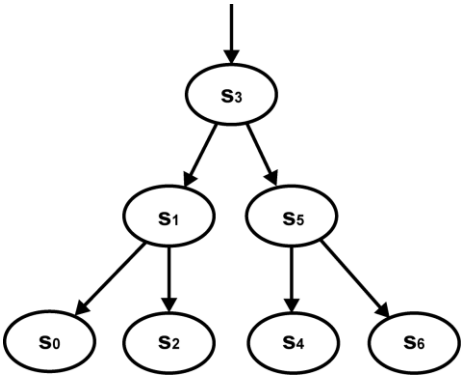
Neinformované metody aplikují jednotlivé operátory slepě, procházejí stavovým prostorem bez jakékoli aprioriní informace o výhodnosti aplikace toho kterého operátoru. Pro daný stav jsou tedy zpravidla všechny přechody (všechny aplikace operátorů) stejně pravděpodobné. Metody nijak nehodnotí, zda je aktuální stav či cesta k němu výhodná a nesnaží se odhadovat, zda je cíli bližší či vzdálenější. Nicméně jsou jednoduché a nevyžadují výpočet žádné složité ohodnocovací funkce.

Oproti tomu informované metody se snaží na základě nějaké jednodušší či složitější funkce odhadnout, zda bude daný přechod z konkrétního stavu z hlediska řešení problému výhodnější než jiný. Pokud pomineme psychologii, postupují tyto algoritmy podobně jako šachista, který se na základě zhodnocení aktuální situace snaží rozhodnout, který následující tah zvolí. Není schopen v té chvíli vyhodnotit všechny možné kombinace (prohledat všechny stavy) na více tahů dopředu, ke svému rozhodnutí tedy používá heuristiku a zvolí tah, který se mu v tu chvíli zdá jako nejvýhodnější. Hráč s lepší heuristikou pak zpravidla vítězí.

Zvláštní třídu představují metody lokální, které mají hlavní výhodu ve své relativně malé paměťové náročnosti a snadné implementaci. V zásadě se jedná o informované metody, které ale vyhodnocují v každém stavu jen výhodnost nejbližších následníků daného stavu a rezignují na zapamatování si stavů předchozích. Chovají se tedy lokálně, nevrací se v orientovaném grafu zpět, postupují stále jen kupředu až do chvíle, kdy jsou všichni následníci méně vhodní, než aktuální uzel. Největším problémem těchto metod tak představuje skutečnost, že mohou uváznout v lokálním minimu.

1.1.3. Hodnocení metod

Pokud chceme zvolit vhodnou metodu prohledávání stavového prostoru, musíme mít možnost tyto metody popsat a porovnávat. K tomu nám slouží základní parametry metod, které je charakterizují. Předpokládejme orientovaný graf, který metoda prohledává ve formě stromu zobrazeného na následujícím obrázku.



Obr. 4 *Prohledávací strom. Uzel je představován příslušným stavem úlohy a dalšími informacemi nutnými pro prohledávání daným algoritmem.*

Zaveďme si následující pojmy:

- Faktor větvení, značíme b.
- Hloubka cíle, značíme d.
- Maximální délka cesty, značíme m.

Faktor větvení b udává, kolik následníků může být průměrně z každého stavu vygenerováno.
V případě hodnocení výkonnosti nějaké heuristiky, používáme i pojem efektivní faktor větvení, označovaný jako b^*. Efektivní faktor větvení udává takovou hodnotu větvení, která by po nalezení cíle v hloubce d a počtu expandovaných stavů N odpovídala rovnoměrně vyváženému stromu hloubky d s počtem stavů (uzlů) N. V případě zcela ideální heuristiky by byl efektivní faktor větvení roven jedné. Hloubka cíle d označuje počet úrovní stromu, kterými musíme stromem od kořene (počáteční stav) k cíli na nejnižší úrovni projít. Hloubka je počítána od kořene, který leží v úrovni 0. Maximální délka cesty odpovídá nejdelší možné cestě ve stavovém prostoru, která je při prohledávání realizována (generována).

U metod prohledávání stavového prostoru se budeme zajímat, zda

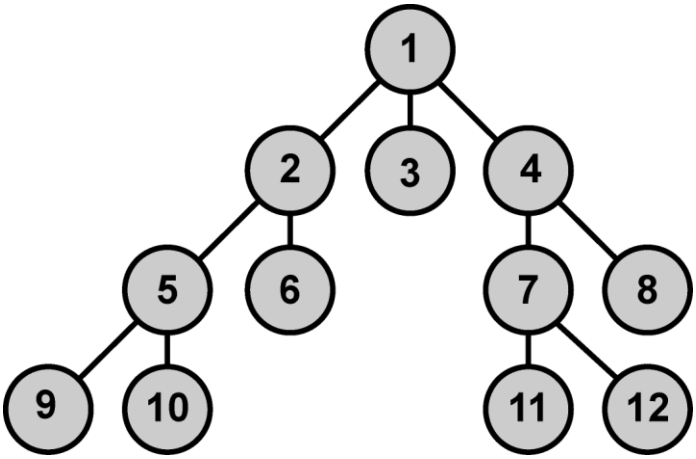
- Je metoda úplná.
- Je optimální.
- Jaká je její časová složitost.
- Jaká je její prostorová složitost.

Říkáme, že metoda je úplná, pokud nalezne řešení vždy, když existuje. Optimální metoda nalezne navíc řešení nejlepší. Nalezne tedy řešení s nejkratší cestou v případě uniformě ohodnocených hran přechodů, respektive řešení s nejnižší cenou těchto přechodů v případě jejich neuniformního ohodnocení.

Časová složitost udává, v kolika krocích, tedy kolika aplikacích operátorů přechodu bude řešení nalezeno. Podobně prostorová složitost říká, kolik paměti, obvykle ve smyslu počtu uchovávaných stavů, bude pro algoritmus zapotřebí.

1.1.4. Neinformované prohledávání

Jak již bylo řečeno, společným rysem neinformovaných metod je, že při pořadí procházení jednotlivých stavů nepoužívají žádnou ohodnocovací heuristickou funkci, která by odhadovala výhodnost následníků daného stavu a vybírala toho nejnadějnějšího z nich. Algoritmy postupně prohledávají vyhledávací strom jen v závislosti na jeho topologii, označujeme je také jako slepé metody prohledávání stavového prostoru.



Obr. 5 *Prohledávací strom - příklad.*

Jednotlivé algoritmy se liší pořadím, ve kterém jsou stavy generovány a tedy i procházeny. Generování cesty můžeme zapsat pomocí syntaxe, kdy začínáme z počátečního stavu 1 [(1)] a postupně generujeme uzly další, které jsou výsledkem provedení operátoru φ na zvolený stav, například [φ(1)] -> [(2,1)(3,1)(4,1)]. Poté je operátor přechodu φ aplikován na další stav z vygenerované množiny. Získáme tak vygenerovanou posloupnost stavů včetně cesty, které do nich vedly. Je třeba si uvědomit, že mezi pojmy uzel a stav je jistý rozdíl. Zatímco pojmem stav rozumíme skutečně jen stav úlohy jako takový, uzly vyhledávacího stromu zahrnují mimo stavu samotného i další informace. Jedná se zpravidla o informaci o cestě k uzlu (odkaz na rodiče), aplikované operátory přechodu, hloubku uzlu ve vyhledávacím stromě. V případě informovaného prohledávání pak i ohodnocení uzlu heuristickou funkcí.

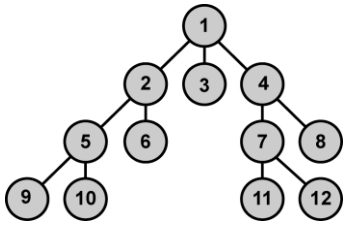
Obvyklé bývá u neinformovaných metod zavedení dvou množin uzlů, množiny OPEN, která obsahuje dosud neprověřené, nově generované uzly a množiny CLOSE, která naopak obsahuje uzly již prověřené, expandované. Uzly obsažené v množině CLOSE již nejsou při expanzi znova zařazovány do množiny OPEN a jsou tedy prověřeny jen 1x.

V následující části se seznámíme s některými neinformovanými metodami prohledávání stavového prostoru a přidržíme se jejich anglického pojmenování.

1.1.1.1. Metoda Breadth-first search (BFS)

Popis

Jinak nazývaná také jako prohledávání do šířky. Spolu s metodou prohledávání do hloubky základní algoritmus prohledávání stavového prostoru. Algoritmus BFS prohledává strom uzlů postupně po jednotlivých úrovních a další úroveň stromu je prohledávána až po prohledání všech uzlů úrovně předcházející.



Obr. 6 *Prohledávací strom - příklad.*

Začátek generování cesty algoritmu BFS lze pro předcházející graf zkráceně demonstrovat následujícím zápisem:

[φ (1)] -> [φ (2,1) (3,1) (4,1)] -> [φ (3,1) (4,1) (5,2,1) (6,2,1)] -> [φ (4,1) (5,2,1) (6,2,1)] -> [φ (5,2,1) (6,2,1)(7,4,1)(8,4,1)] -> …

Generování této cesty odpovídá principu fronty FIFO (first in, first out), kdy je vždy vybrán první, nejlevější uzel z fronty OPEN, ten je prověřen a v případě, že nepatří mezi cílové stavy je expandován. Nově vygenerované uzly jsou zařazeny na konec fronty. Algoritmus pracuje tak dlouho, dokud nenarazí na první cílový stav, nebo dokud nevyčerpá celý stavový prostor – množina OPEN je prázdná.

Vlastnosti algoritmu

Algoritmus je úplný pro konečný faktor větvení b. Je také optimální z pohledu délky cesty.

Časová složitost algoritmu je exponenciální, kdy počet navštívených uzlů pro cíl g v hloubce d můžeme pro faktor větvení b vyjádřit jako

$O(BFS) = 1 + b + b^2 + \ldots + b^d + b(b^d-1) = (b^{d+1})$

Nejprve procházíme uzly všech úrovní předcházející úrovni, ve které se nachází cíl. To odpovídá první součtové části výrazu. Pokud je cíl ve hloubce d a navíc v pesimistickém případě na konci této úrovně, pak musíme ještě prověřit všechny vrstvy na úrovni, kde se cíl nachází a to včetně jeho samotného – odpovídá hodnotě b^d uzlů ve výrazu. Navíc ale musíme pro všechny uzly v této vrstvě, mimo stavu cílového, aplikovat dle algoritmu BFS operátor φ na všechny uzly této vrstvy. Tuto skutečnost postihuje poslední část výrazu O(BFS), tedy $b(b^d-1)$.

Prostorová složitost je stejná jako časová, všechny uzly musíme udržovat v paměti.

$O(BFS) = (b^{d+1})$

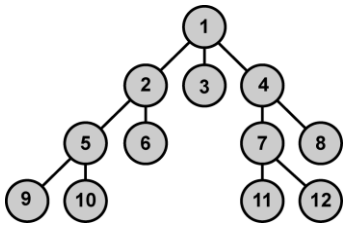
Výhody a nevýhody

Hlavní výhodou je v případě předpokládaného uniformního ohodnocení hran optimálnost a úplnost algoritmu. Za tyto výhody však algoritmus platí velkou paměťovou složitostí, je tedy pro rozsáhlé prostory prakticky obtížně použitelný, neboť může při implementaci snadno vyčerpat fyzicky dostupné paměťové zdroje počítače. Časová složitost je v porovnání s jinými algoritmy jen průměrná.

1.1.1.2. Metoda Depth-first search (DFS)

Popis

Metoda patří spolu s BFS mezi základní prohledávací algoritmy. Jinak nazývaná také jako prohledávání do hloubky. Algoritmus DFS prohledává strom uzlů tak, že se nejlevější větví stromu postupně zanořuje do maximální možné hloubky, tedy do konce této větve, nebo do nalezení cíle. Poté se vrací o jeden krok zpět a provádí analogicky zanoření pro další větev aktuálně zpracovávaného uzlu. Neprochází tedy strom postupně po jednotlivých úrovních jako BFS, ale rekurzivně po jednotlivých podstromech daného uzlu. Zjednodušeně lze chápat tuto prohledávací metodu jako rekurzivní volání funkce „Vyhodnoť“ – vyhodnoť, zda je zpracovávaný uzel cílem a když není, vyhodnoť tuto podmínku pro jeho následníka zleva.



***Obr. 7** Prohledávací strom - příklad.*

Začátek generování cesty algoritmu DFS lze pro předcházející graf zkráceně demonstrovat následujícím zápisem:

[φ (1)] -> [φ (2,1) (3,1) (4,1)] -> [φ (5,2,1) (6,2,1) (3,1) (4,1)] -> [φ (9,5,2,1)(10,5,2,1) (6,2,1) (3,1) (4,1)] -> [φ (10,5,2,1) (6,2,1) (3,1) (4,1)] -> [φ (6,2,1) (3,1) (4,1)] -> [φ (3,1) (4,1)] -> [φ (4,1)] -> [φ (7,4,1) (8,4,1)] …

Generování této cesty odpovídá principu zásobníku LIFO (last in, first out), kdy je vždy vybrán první, nejlevější uzel z množiny OPEN, ten je prověřen a v případě, že nepatří mezi cílové stavy je expandován. Nově vygenerované uzly jsou zařazeny na začátek množiny, vrchol zásobníku. Algoritmus pracuje tak dlouho, dokud nenarazí na první cílový stav, nebo dokud nevyčerpá celý stavový prostor – množina OPEN je prázdná.

Vlastnosti algoritmu

Pro obecně definovaný strom není algoritmus úplný, neboť může uváznout v nekonečně dlouhé větvi. Není optimální, nalezené řešení nemusí být řešením ležícím v minimální hloubce.

Časová složitost algoritmu je stejně jako pro BFS exponenciální, kdy počet navštívených uzlů odpovídá maximální hloubce zanoření m a pro faktor větvení b ji můžeme vyjádřit jako

O(DFS) = (b^{m+1})

Prostorová složitost je lineární, neboť vždy udržujeme v paměti jen aktuálně prohledávanou větev.

O(DFS) = (bm)

Výhody a nevýhody

Hlavní výhodou je dobrá prostorová složitost, která odpovídá nejdelší větvi v grafu a tedy relativně malé nároky na paměť počítače při implementaci algoritmu. Nevýhodou je naopak, že metoda není pro nekonečné obecné stromy úplná. Tuto limitaci lze ale relativně snadno potlačit modifikací metody, jak si ukážeme dále. Metoda také není optimální, což může být v případě úloh, kdy hledáme pouze nejlepší možné řešení důvodem pro nepoužití této metody.

Časová složitost je opět jen průměrná. K časové i prostorové složitosti je ale třeba poznamenat a uvědomit si, že v případě řešení reálné úlohy může být hloubka d nalezeného cíle mnohem menší než maximální hloubka m a algoritmus tak není nucený ve všech případech nejdelší větev skutečně expandovat.

1.1.1.3. Metoda Limited depth-first search (LDFS)

Popis

Jedná se metodu principiálně naprosto shodnou s DFS, pouze je zde definována maximální hloubka povoleného zanoření algoritmu l.

Vlastnosti algoritmu

Algoritmus není úplný pro cíl ležící v hloubce d vetší než je maximální povolená hloubka zanoření l. Pro l<d(g) tedy není úplný. Jinak ano. Optimální není.

Časová i prostorová složitost algoritmu odpovídá DFS s limitní hloubkou l, tedy časová složitost LDFS je

O(LDFS) = (b^{l+1})

Prostorová složitost LDFS je rovna

O(LDFS) = (bl)

Výhody a nevýhody

Jsou opět stejné, jako v případě DFS, LDFS pouze zajišťuje ukončení běhu algoritmu v konečném čase a nalezení řešení, které je v menší, než limitní hloubce.

1.1.1.4. Metoda Iterative depth-first search (IDFS)

Popis

Nazývána také metodou prohledávání stavového prostoru s postupným prohledáváním. Vychází z algoritmu DFS, respektive LDSF. Modifikace je taková, že parametr l, který udával v metodě LDFS maximální možnou hloubku zanoření není v IDFS konstantní, ale nabývá postupně hodnot od l = 1,2…n. Jde tedy o opakované spouštění prostého algoritmu LDSF, kdy prohledávání LDFS je

spouštěno vždy od začátku pro každý postupně narůstající parametr l až do chvíle nalezení cíle či vyčerpání stavového prostoru.

Vlastnosti algoritmu

Pro konečné větvení b je tento algoritmus úplný. Je i optimální.

Časová složitost je exponenciální, podobně jako u BFS a DFS.

$O(IDFS) = d(b^1) + (d-1)b^2 + \ldots + 1(b^d) \approx (b^m)$

Zápis časové složitosti vlastně odpovídá tomu, že při opakovaném spouštění algoritmu DFS projdeme pro cíl v hloubce d d-krát uzly první úrovně stromu, (d-1)-krát uzly další úrovně atd. Algoritmus DFS je tedy spuštěn d-krát vždy s maximální hloubkou l o jedničku větší. Reálně je často časová složitost metody IDFS lepší, než u BFS, která přidává ještě expanzi jedné vrstvy uzlů (na =úrovni stromu obsahující cíl).

Prostorová složitost je lineární, stejná jako u DFS

$O(IDFS) = (bm)$

Výhody a nevýhody

Jedná se o neinformovanou metodu první volby, pokud neznáme velikost a maximální hloubku prohledávaného prostoru. Kombinuje v sobě výhody metod DFS a BFS, tedy relativně nízké paměťové nároky a zejména úplnost a optimálnost. Časová složitost je jen průměrná. V praktické úloze horší než DFS, ale často lepší než BFS. Například pro b = 10 a d = 5 dostáváme pro časovou složitost

O(IDFS) = 50 + 400 + 3 000 + 20 000 + 100 000 = 123 450

O(BFS) = 10 + 100 + 1 000 + 10 000 + 100 000 + 999 990 = 1 111 100

1.1.1.5. Metoda Uniform cost search (UCS)

Popis

Jedná se o modifikaci algoritmu BFS. Tuto metodu řadíme také mezi neinformované metody, byť pro ni neplatí, že přechody mezi jednotlivými stavy jsou stejně pravděpodobné. Hrany jednotlivých přechodů v grafu jsou oproti BFS ohodnoceny různými, známými hodnotami, které vyjadřují náročnost, cenu přechodu při aplikaci daného operátoru. Procházené uzly následníků jsou uspořádávány ve frontě s prioritou podle nižší ceny přechodu k následníkovi a v tomto pořadí tedy i procházeny.

Vlastnosti algoritmu

Vlastnosti algoritmu opět odpovídají algoritmu BFS. Algoritmus je úplný a optimální pro nenulovou cenu přechodů větší než ε, kde ε je konstanta.
Časová i prostorová složitost je rovna

$O(UCS) = O(b^{1+[C^*/\varepsilon]})$, kde C* je cena optimálního řešení.

Výhody a nevýhody

Jsou naprosto shodné jako u výchozího algoritmu BFS.

1.1.1.6. Souhrn neinformovaných metod

Preferovaným algoritmem neinformovaného prohledávání je vzhledem ke své univerzálnosti algoritmus IDFS. Souhrn vlastností popsaných algoritmů je uveden v následující tabulce.

Vlastnost	BFS	UCF	DFS	LDFS	IDFS
úplnost	Ano	Ano	Ne	Ano, l>=d	Ano
optimálnost	Ano	Ano	Ne	Ne	Ano
časová složitost	$O(b^{m+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^{m+1})$	$O(b^l)$	$\approx O(b^m)$
prostorová složitost	$O(b^{m+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(bl)$	$O(bm)$

Tab. 1 *Souhrn vlastností neinformovaných metod prohledávání.*

1.1.5. Informované heuristické prohledávání

Přestože neinformované metody prohledávání jsou při hledání cílů úspěšné, pro velké stavové prostory není jejich použití ani v optimalizovaných variantách s ohledem na časovou a prostorovou složitost použitelné. Tyto algoritmy řešícími prohledávání stavového prostoru hrubou silou se tedy snažíme nahradit algoritmy „chytřejšími“. Chceme tedy nalézt metodu, která by využila nějakou další znalost o řešeném problému a na základě ní použila heuristickou metodu, která povede s vysokou nadějí k rychlejšímu nalezení hledané cesty s co nejmenšími prostředky. Metody se snaží tedy postupovat na základě nějaké heuristiky podobně jako lidé, kteří například zabloudí v lese a hledají z něj cestu ven. Mohou postupovat podle různých algoritmů, například vždy směřovat tam, kde jsou stromy nejméně husté, sledovat vodní toky atd. Zdaleka nemohou vědět, že tato cesta vede k cíli či dokonce zda je nejkratší, nicméně s vysokou mírou pravděpodobnosti je vhodná heuristika nakonec dovede k cíli.

V případě informovaného prohledávání tedy opět hledáme cestu ze stavu počátečního do stavů cílových. Ke generování potencionální cesty k cíli však využíváme nějakou další apriorní informaci o řešené úloze. Zavádíme ohodnocení každého n-tého uzlu funkcí

$f(n)=h(n) + g(n)$

, kde $h(n)$ je odhad ceny cesty z n-tého uzlu do cíle a $g(n)$ představuje zpravidla cenu cesty z počátečního stavu do aktuálního n-tého uzlu.

Pořadí prohledávaných uzlů je pak určeno podle hodnoty funkce $f(n)$ tak, aby byly nejdříve prověřeny nadějné stavy s nejmenší hodnotou $f(n)$. Hodnotu funkce $g(n)$ je možné vyjádřit pro aktuální stav n přesně, představuje ohodnocení již absolvované cesty. Funkce $h(n)$, která je samozřejmě také vypočtena přesně, představuje jen odhad ceny cesty do cílového stavu. Čím nižší je hodnota $h(n)$, tím blíže se daný stav pravděpodobně nachází k cíli. Funkce $h(n)$ je tedy odhadem, metodu jejího výpočtu určuje člověk na základě svých zkušeností a znalostí řešeného problému. Na kvalitě heuristické funkce pak přímo závisí efektivita nalezení řešení.

Volba optimální heuristické funkce nemusí být triviální. Zpravidla při jejím hledání postupujeme metodou relaxace problému. Relaxace problému představuje mechanismus, jak nalézt přípustnou heuristiku. Za přípustnou (optimistickou) heuristiku považujeme takovou, která nenadhodnocuje cenu cesty k cíli, tedy reálné ohodnocení ceny cesty do cíle bude vždy vyšší, než je výsledek heuristické funkce.

Relaxace problému spočívá v jeho podstatném zjednodušení, tedy vlastně ignorujeme jistá omezení úlohy, která činí problém složitější. Nalezená heuristika ale nemusí být zdaleka optimální a nemusí tedy vést k podstatnému zlepšení vlastností prohledávacího algoritmu. Nalezení dobré heuristické funkce je tedy pro efektivitu řešení velmi podstatné. Ukažme si vše opět na příkladu hlavolamu „Loydova patnáctka“, kdy máme za úkol v matici 4x4 prvky seřadit 15 očíslovaných kamenů pomocí jejich přesunu s využitím jednoho volného pole. Zvažujeme, který kámen přesunout a do kterého následného stavu tedy přejít. Chceme určit heuristiku h(n), která bude odhadovat, jak je daný stav vzdálen od cíle. Možností je více - můžeme například pro každý stav vyčíslit počet chybně rozložených kamenů, nebo vyčíslit sumu vzdáleností kamenů od jejich korektních pozic.
V následujícím tahu pak můžeme volit takový přesun kamene, který povede do stavu s minimální hodnotou této funkce, jak si ukážeme následně u metody Greedy search.

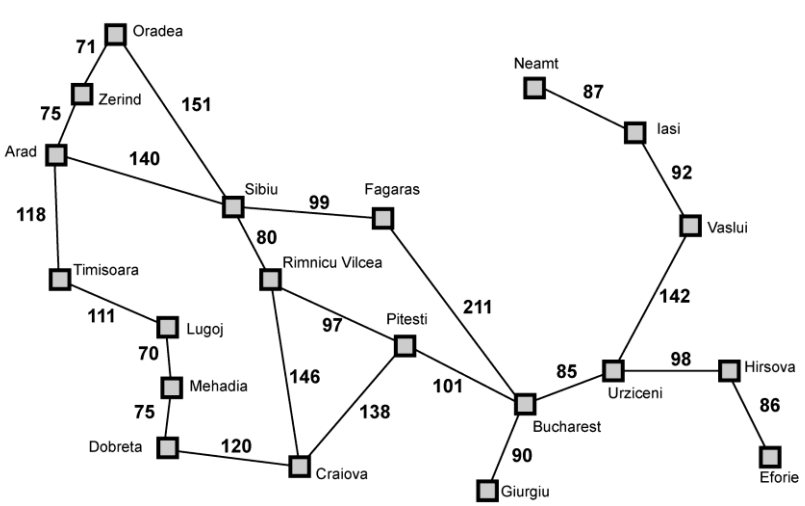
1.1.1.7. Metoda Gready search (GS)

Popis

Algoritmus představuje nejjednodušší variantu informovaného prohledávání. Bývá také nazýván hladovým algoritmem. Nesleduje cenu cesty do daného uzlu, funkce g(n) = 0. Ohodnocovací funkce f(n) se tedy redukuje pouze na odhad vzdálenosti z daného n-tého uzlu do cíle, f(n)=h(n).

Algoritmus preferuje uzly s nejmenší odhadnutou vzdáleností do cíle. Expanduje tedy vždy uzel, který se dle funkce f(n) zdá nejbližší k cíli.

Algoritmus si budeme demonstrovat na klasickém příkladu hledání cesty na mapě Rumunských měst.



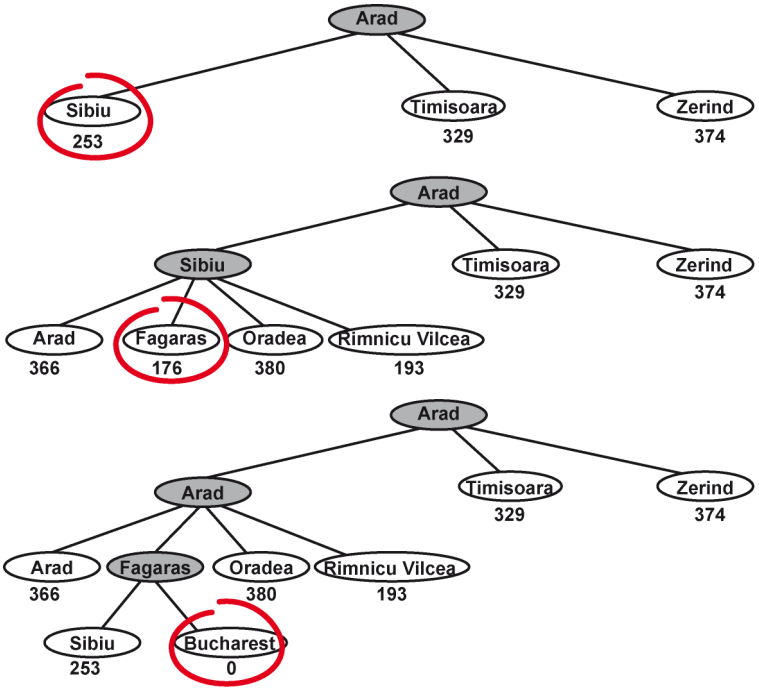
Vzdálenost přímou linkou do Bukurešti	
Arad	**366**
Bucharest	**0**
Craiova	**160**
Dobreta	**242**
Eforie	**161**
Fagaras	**178**
Giurgiu	**77**
Hirsova	**151**
Iasi	**226**
Lugoj	**244**
Mehadia	**241**
Neamt	**234**
Oradea	**380**
Pitesti	**98**
Rimnicu Vilcea	**193**
Sibiu	**253**
Timisoara	**329**
Urziceni	**80**
Vaslui	**199**
Zerind	**374**

Obr. 8 *Schematická mapa Rumunských měst.*

Máme k dispozici mapu měst a silnic, které je propojují, včetně délky těchto silnic.

Dále máme k dispozici tabulku, která udává vzdušnou vzdálenost každého města od Bucharesti. Chceme, aby algoritmus nalezl cestu z Aradu (počáteční stav) do cíle v Bucharesti (cílový stav). Algoritmus se tedy musí rozhodnout, zda se z Aradu vydá do Timisoary, Sibiu, nebo Zerindu.

Uvědomme si, že algoritmus nevidí celou mapu, má k dispozici jen tabulku vzdušných vzdáleností. Algoritmus tedy v následujícím kroku zvolí to město, které je dle vzdušné vzdálenosti (heuristiky) k cíli nejblíže. Zde to tedy bude Sibiu. Dále algoritmus postupuje stejně, tedy ze Sibiu se vydá Fagarase a z něj už do cíle v Bucharesti. Algoritmus tedy vždy hledá lokální minimum heuristické funkce a doufá, že dorazí do minima globálního.



Obr. 9 *Postupné generování cesty – hladový algoritmus*

Algoritmus byl tedy úspěšný a pomocí této heuristiky směřoval do cíle efektivněji, než v případě neinformovaného hledání, které by prověřovalo postupně celou mapu jen na základě její topologie.

Nalezená cesta ale není optimální, má délku 450km, zatímco nejkratší cesta pouze 418km. Hladový algoritmus se od optimálního řešení odchýlil ve městě Sibiu, kde zvolil hladově jako další uzel Fagaras místo celkově optimálnější cesty přes Rimnicu Vilcea. Hladové prohledávání se tedy snaží vždy ukousnout co největší kus zbývající cesty, bez ohledu na její celkovou délku. Nicméně tato strategie bývá nejen v případě prohledávání stavového prostoru relativně úspěšnou strategií vedoucí k vítězství.

Vlastnosti algoritmu

Algoritmus není úplný. Není ani optimální. Časová i prostorová složitost odpovídá neinformovanému prohledávání, tedy

$O(GS) = (b^m)$.

Je třeba si ale uvědomit, že použitá heuristika dokáže reálně dobu prohledávání často významně zredukovat. Redukci času potřebného k nalezení cíle však nelze zaručit.

Výhody a nevýhody

Výhoda algoritmu je vlastně jediná – jde o informovaný algoritmus využívající heuristickou funkci, která dokáže významně redukovat reálnou časovou náročnost nalezení řešení. Nicméně není úplný, ani optimální a je zde nebezpečí uváznutí algoritmu v lokálním minimu. Představme si například úlohu hledání cesty z Neamt do Fagaras. Metodou hladového prohledávání expandujeme z Neamt nejprve jediný možný uzel, Lasi. Po vyhodnocení heuristiky pro obě sousední města, tedy Vasluj a Neamt zjistíme, že vzdušná čára z Neamt do Fagaras je kratší, než vzdušná vdálenost z Vasluj. Heuristika by algoritmus tedy opět vrátila do uzlu Neamt a tak stále dále. Proto je vhodné zavést například detekci opakovaných stavů a zamezit tak podobnému zacyklení. Algoritmus má také poměrně velkou prostorovou složitost.

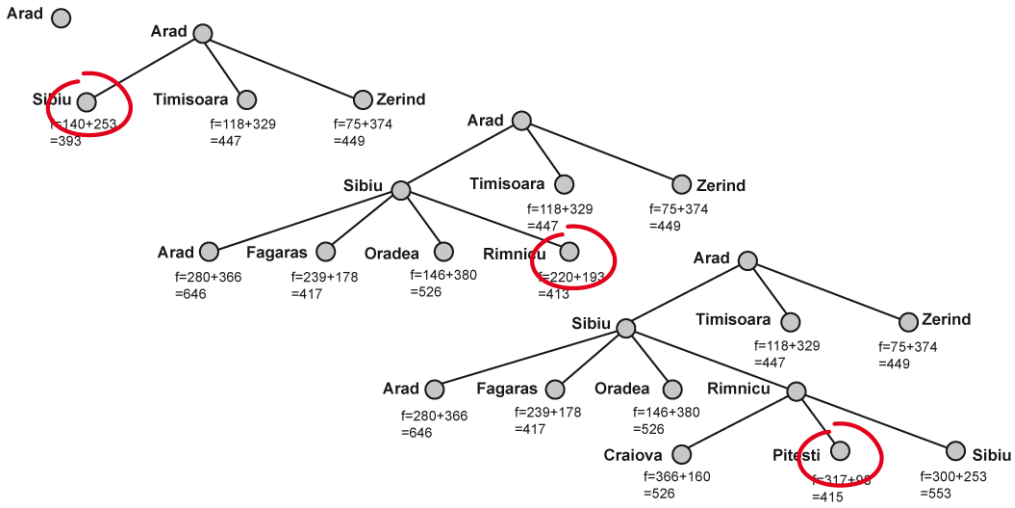
*1.1.1.8. Metoda A**

Popis

Algoritmus využívá opět principu hladového prohledávání, ale do vyhodnocení heuristické funkce již zahrnuje i cenu cesty do aktuálního n-tého uzlu, tedy f(n)=h(n) + g(n). Funkce g(n) je cena cesty do uzlu n, h(n) odhad ceny cesty z n-tého uzlu do cíle g. Funkce f(n) tedy představuje odhad nejlevnější cesty do cíle g, která vede přes n.

Algoritmus vyžaduje použití přípustné heuristiky $0 \leq h(n) \leq h^*(n)$, kde $h^*(n)$ je skutečná cena cesty do cíle. Heuristická funkce tedy provádí optimistický odhad ceny cest do cíle, žádná reálná cesta nemůže mít nižší cenu (být kratší), než použitá heuristika. V našem příkladě rumunských měst je toto splněno, neboť vzdušná vzdálenost mezi dvěma městy musí být vždy kratší nebo limitně stejná, jako délka silnice, která je spojuje.
Principiálně funguje algoritmus A* naprosto stejně jako prohledávání do šířky, pouze s tím rozdílem, že provádí prioritní seřazení stavů podle hodnoty funkce f(n).

Zkusme se podívat, jak by se algoritmus A* choval při nalezení cesty z Aradu du Bucharesti.



***Obr. 10** Postupné generování cesty – A* algoritmus*

Až do uzlu Sibiu algoritmus postupuje stejnou cestou, jako GS. Zde vyhodnotí funkci f(n) pro všechny následníky. Přesto, že Fagaras je oproti Rimnicu vzdušnou čarou blíže k cíli (178 km oproti 193 km), metoda A* zvolí jako další expandovaný uzel Rimnicu. Po započtení ceny cesty z Aradu do Fagarase (239 km) resp. Rimnicu (220 km) totiž dochází algoritmus k tomu, že cesta přes Rimnicu (413 km) je celkově výhodnější oproti cestě přes Fagaras (417 km). Hodnota funkce f(n) je stále jen odhadem délky cesty, neboť mimo exaktně vyčíslené ceny (délky) již absolvované cesty obsahuje i odhad délky cesty následující založený jen na vzdušné vzdálenosti mezi městy. Nicméně přípustnost heuristiky zaručuje nalezení optimální cesty.

Vlastnosti algoritmu

Algoritmus A* je za předpokladu konečného počtu uzlů úplný. Pro ohodnocení uzlů funkcí f(n) < C*, kde C* je cena cesty do cíle, expanduje A* všechny uzly s f(n) < C*, některé s f(n) = C* a žádné s f(n) > C*. Algoritmus A* je optimální za předpokladu použití přípustné heuristiky.

Časová složitost algoritmu je

$O(A^*) = (b^m)$.

ale heuristika dokáže reálně dobu prohledávání často podstatně redukovat.

Prostorová složitost je v základní variantě rovna

$O(A^*) = (b^m)$.

Je ale možné použít modifikace pro její redukci, jako například použití algoritmu IDS podle maximální ceny l, kdy pro každou další iteraci IDS volíme jako počáteční uzel s nejnižší hodnotou funkce f(n), která je ale současně vyšší než aktuální l. Tato metoda je označována jako IDA*. Další možností je pak ořezávat neslibné stavy a cesty do těchto stavů. Ušetříme tak kapacitu paměti za cenu suboptimálnosti takového řešení.

Výhody a nevýhody

Jedná se o úspěšný algoritmus, kdy samozřejmě v závislosti na kvalitě heuristiky obvykle významně redukuje časovou složitost prohledávání. Navíc je úplný a optimální, neprodlužuje již dlouhé cesty. Podobně jako v případě GS je zapotřebí ošetřit možnost zacyklení algoritmu. Relativně velkou paměťovou složitost lze redukovat za cenu suboptimálnosti algoritmu.

1.1.6. Lokální metody prohledávání

Samostatnou oblast metod pro prohledávání stavového prostoru jsou takzvané lokální metody prohledávání. Zmiňme si stručně alespoň dvě z nich a to Hill climbing (slézání kopce, gradientní metoda prohledávání) a Simulated annealing (simulované žíhání). Společným znakem je, že tyto metody vyhodnocují jen své nejbližší okolí a vydají se při prohledávání zkrátka některým směrem, který se v tu chvíli zdá metodě lokálně optimální na základě vyhodnocení funkce f(n). Chovají se tedy podobně, jako GS. Lokální metody ale zcela zapomínají předcházející uzly a postrádají tak možnost návratu.

Algoritmus simulovaného žíhání se oproti jednoduchému gradientnímu prohledávání liší v tom, že zavádí pojem teploty T. Tato teplota je na počátku prohledávání vysoká a v jeho průběhu postupně

klesá až k nule. Při výběru následníka není v každém uzlu vždy zvolen jen následník nejvýhodnější, jako v případě gradientního prohledávání, ale s jistou pravděpodobností, která závisí na teplotě T, je zvolen přechod do uzlu jiného. S klesající teplotou pravděpodobnost volby neoptimálního následníka klesá, algoritmus postupně konverguje do ustáleného stavu. Algoritmus s větší pravděpodobností dosáhne globálního minima (cíle). Zavedením teploty je omezena možnost uváznutí algoritmu v lokálním minimu, protože s jistou pravděpodobností se z něj dokáže algoritmus vymanit a prohledat tak i ty části stavového prostoru, kam by prostá gradientní metoda nevstoupila. Často je tento algoritmus při neúspěchu startován opakovaně z rozdílných počátečních míst stavového prostoru.

Výhodami lokálních algoritmů je zejména jejich snadná implementace a minimální paměťové nároky. Nevýhodou je především jejich suboptimálnost a neúplnost, mohou snadno uváznout v lokálním minimu. Přesto jsou pro řadu problémů úspěšně využívány.

1.6. Seznam použité literatury

[1] Kubalík, J., Základy umělé inteligence, stránky předmětu, https://cw.felk.cvut.cz/wiki/courses/y33zui/program

[2] Barták, R., Umělá inteligence I, stránky předmětu http://ktiml.mff.cuni.cz/~bartak

[3] Pavliš, M., Algoritmy pro hledávání cesty, Bakalářská práce, 2007, http://autnt.fme.vutbr.cz/szz/2007/BP_Pavlis.pdf

[4] Berka, P., Úvod do umělé intelligence, stránky předmětu, http://sorry.vse.cz/~berka/4IZ229/

[5] http://cs.wikipedia.org/wiki/Patn%C3%A1ctka