Jak nám heuristiky usnadňují řešení problémů?

Y. Herashchanka[1], J. Lezna[2], and M. Richter[3]

[1]yherashchanka@gmail.com, Gymnázium Turnov
[2]josef.lezna@gymzn.cz, Gymnázium doktora Karla Polesného Znojmo
[3]2018-richter-matous@gymtan.cz, Gymnázium Tanvald

Abstrakt

V tomto článku budou představeny pojmy „*stavový prostor*“ a „*inteligentní agent*“. Následně bude čtenář seznámen s některými nejdůležitějšími vyhledávacími algoritmy pro stavový prostor. Bude definován pojem „*heuristika*“ a následně ukázáno, že algoritmy využívající heuristiky si povedou obecně nejlépe.

1 Teoretický základ

V rámci miniprojektu jsme se zabývali algoritmy pro řešení problémů ve stavovém prostoru.

Stavový prostor je formalizací určitého problému, nejčastěji realizovaný pomocí grafu; jeho vrcholy pak představují různé stavy, které v rámci řešení nebo časového vývoje problému mohou nastat; (orientované) hrany přechody mezi těmito stavy. Hranám je možno přiřazovat parametry, jako například „cenu“ daného přechodu (hranové ohodnocení).

V takovémto stavovém prostoru uvažujeme entitu zvanou inteligentní agent, jejíž akce způsobují přechody mezi stavy. Pro naše potřeby takovýto popis postačí. V informatické praxi i teorii se pojem rozšiřuje mimo stavové prostory a často se jedná o umělou inteligenci v reálném i simulovaném prostředí.

Pro porozumění dalšího textu je důležitá znalost pojmu „heuristika“. V kontextu informatiky jde o metodu, jak *rychle* najít *přibližně* nejlepší řešení (v kontrastu s globálně optimálním řešením, jehož nalezení si může vyžádat čas nad rámec lidských možností). V případě našich vyhledávacích algoritmů spočívá heuristika v nějaké metodě odhadu, který další krok povede k nejlevnější cestě (a který by tedy bylo nejlépe zahrnout v dalších úvahách); tedy vlastně v hodnocení nabízejících se možností.

Vlastnosti jednotlivých algoritmů jsme porovnávali ve stavovém prostoru vytvořeném abstrakcí myšlené mapy pseudoskutečného terénu. Podrobněji v sekci 3.

2 Způsoby řešení

Nejprve uvedeme obecný algoritmus pro prohledávání grafů [1]. Uvažujme graf G s vrcholy V, které rozdělíme do tří disjunktních podmnožin E (již prozkoumané), F (hraniční) a U (ještě nikdy nenavštívené). Označme s výchozí a t koncový vrchol.

Listing 1: Grafové prohledávání

```
E = set() # mnozina prozkoumanych vrcholu je prazdna
F = {s}   # v hranici lezi pouze vychozi vrchol
while(True)
        if(!F) : return None #nema reseni
        v=vyberZHranice(F)
        if(v==t): return vytvorReseni(v) #nasli jsme cestu
        E=E.add(v) #vrchol je pridan k prozkoumanym
        #doplni sousedy v do hranice, oznaci u nich,
        #ze jsme do nich vstoupili z v a spocita pro ne cenu
        F=aktualizujHranici(F,v)
```

Ve skutečnosti se jedná o celou rodinu algoritmů, konrétní algoritmy odvodíme definováním funkce pro výběr z hranice. Níže uvádíme několi možností výběrové funkce.

1. UCS hledá z každého bodu nejkratší cestu do dalšího bodu. Má vysokou paměťovou složitost, protože si ukládá všechny cesty, které neskončily ve slepé uličce.

2. Breadth First Search (prohledávání do šířky)

   Staový prostor prohledává ve „vlnách“ – za přítomnosti více neprozkoumaných cest se vydá všemi, přičemž si je ukládá. Stačí však mít jen jedno počítadlo kroků. Jakmile nějaká z cest dojde do cíle, je jisté, že se jedná o nejkratší cestu, protože všechny kroky probíhaly paralelně.

3. Greedy Best First Search (hladový algoritmus)

   Algoritmus vyhodnocuje vzdálenost do cíle a vždy si vybírá cestu, která nás nejvíce přiblíží k cíli, nebere ohledy na náklady cesty.

4. Depth First Search (prohledávání do hloubky)

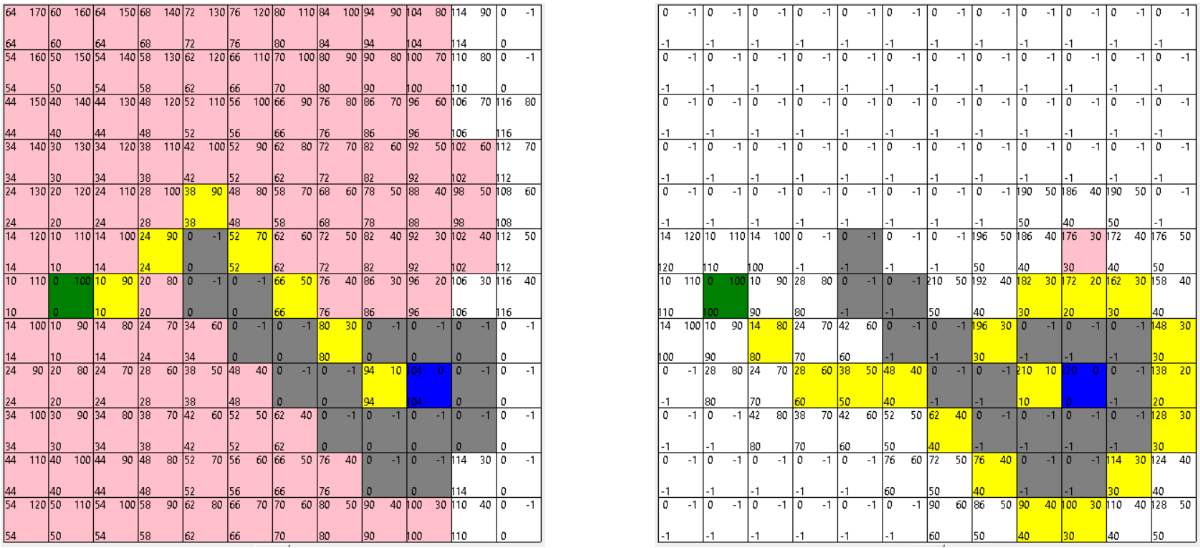
   Vybere si jednu cestu dle nějakého pravidla (třeba pravé ruky) a následuje ji. Pokud skončí ve slepé uličce, vrátí se na poslední křižovatku a prohledává dál. Skončí v cíli a vrátí jedinou uloženou cestu.

5. Algoritmus A* (čti [Éj Stár])

   Jedná se o kombinace GBFS a UCS – vybírá si nejlevnější cestu, která nás v nějakém smyslu nejvíce přiblíží do cíle. Zavádí funkci f – hodnota políčka; $f = g + h$, kde g je cena cesty, h je odhad vzdálenosti do cíle neboli *heuristika*. Podotkněme, že pokud heuristika nepřekročí skutečné náklady, nalezne A* algoritmus optimální řešení.

3 Výsledky experimentu

Pro zjednodušení zadání jsme mapu diskretizovali (rozdělili na čtverečky). Diagonální pohyb by měl normálně mít délku $\sqrt{2}$). Počítače však dokážou daleko rychleji pracovat v celočíselné aritmetice, takže přímou cestu nastavíme na 10, diagonální na 14. Pro odhad vzdálenosti mezi body použijeme manhattanskou vzdálenost, která je definována jako součet absolutní hodnoty rozdílu X-ových souřadnic bodů a absolutní hodnoty rozdílu Y-ových souřadnic bodů.

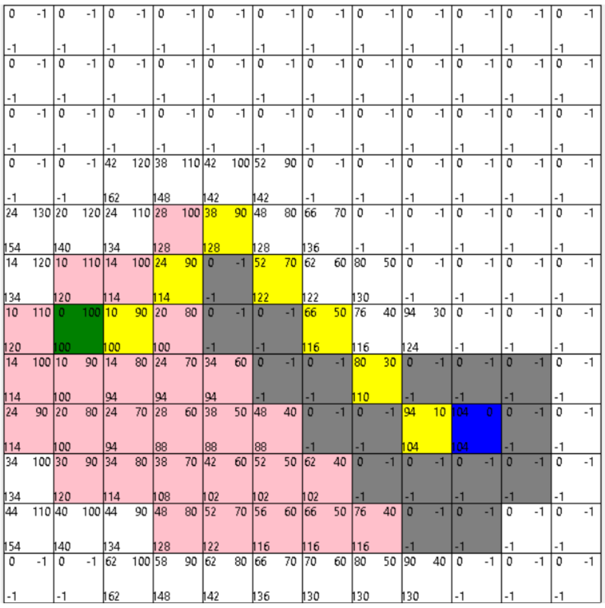


(a) Do šířky (b) Hladový algoritmus

Obrázek 1: Porovnání prohledávání do šířky a hladovým algoritmem

Pro testy jsme napsali skript v Pythonu, ve kterém jsme porovnávali chování jednotlivých výběrových funkcí.

Na obrázku 1a můžeme vidět, že BFS prošel většinu mapy, než našel cestu do cíle. Nalezená cesta je však nejlepší možná.



Obrázek 2: Výsledek prohledávání algoritmem A*

GBFS (viz obrázek 1b) prošel jen malý kus mapy, vhodně navržená překážka však ukazuje jeho nedokonalost, jelikož nejblíže se v daném okamžiku k cíli dostane u okraje „spirály“, proto ji také celou obejde a nejlepší cestu nenajde.

A* na obrázku 2 prošel také značný kus mapy, avšak daleko menší než BFS a nalezl nejlepší cestu, protože hledal kompromis mezi délou trasy a vzdáleností od cíle.

4 Diskuse

4.1 Rozdílnost terénu a nepřátelé

Ve většině her nejsou pouze terény typu „sem nemůžeš“, ale dosti často jsou zde terény „můžeš sem, ale budeš se pohybovat pomaleji, či rychleji“, např. jít do kopce, z kopce, v bažině, na ledě atd. Pro agenta se zde budou lišit hodnoty funkce g, která určuje kolik stojí přesun na další bod. Např. cesta povede do kopce a hodnota přesunu bude větší, nebo z kopce a g bude nižší. Agent se snaží najít nejrychlejší cestu, takže když mu v cestě k cíli nestojí neprostupná překážka ale hora, tak se musí rozhodnout, zda je výhodnější jít přes tuto horu, či ji obejít. S tímto nám velmi pomáhají heuristiky. Bez nich by se agent automaticky rozhodl na prostupnou překážku nejít, protože má větší hodnotu g, ale nějakým způsobem ji obejít, ačkoliv výhodnější cesta by byla rovnou přes ni. Podobným způsobem by bylo možné vypořádat se s nepřátelskými jednotkami. V podstatě se chovají jako pohyblivé překážky a bylo by možné je zahrnout do heuristiky (penalizovat pozice v blízkosti nepřátel).

Obecně se ale dá říci, že algoritmus, který využívá heuristiky je v překonávání překonatelných překážek efektivnější než ten bez nich.

5 Závěr

A jak nám tedy heuristiky usnadňují řešení problémů? Heuristiky nám zužují výběr možností na ty, které mají největší šanci vést k cíli (když jedeme z Prahy do Brna nemusíme brát v úvahu možnost cesty přes Liberec). To nám může řádově zrychlit proces hledání řešení určitého problému. Algoritmus A* je obecně mnohem efektivnější než ostatní algoritmy.

Poděkování

Děkuji jménem nás tří Vojtěchu Svobodovi za to, že pořádá Týden vědy na Jaderce, díky kterému jsme se mohli seznámit s tímto tématem. Také chceme poděkovat Vladimíru Jarému za vedení našeho miniprojektu a za pomoc s prací na sborníkovém příspěvku a na prezentaci.

Reference

[1] S. Russel; P. Norvig. *Artificial Intelligence: A Modern Approach. 2. vyd.* New Jersey, USA: Prentice Hall, 2003. ISBN 0-13-790395-2.

[2] M. TUREČEK. *Demonstrace metod prohledávání stavového prostoru.* Brno, 2010. Bakalářská práce. Vysoké učení technické v Brně. Vedoucí práce František Zbořil.

[3] N. Switft *Easy A* (star) Pathfinding.* [online] 2017 [cit. 20. 6. 2023]. Dostupné z https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2