Algoritmy Cooperative pathfinding (princip a praktické užití)

Kristýna Zemková

1 Úvod

Problém cooperative pathfinding (dále jen CPF) sestává z nacházení nekolidujících cest vedoucích časoprostorem pro agenty, kteří se potřebují přemístit z daného výchozího bodu do předem daného bodu cílového. V tomto příspěvku se pokusím ukázat stávající algoritmy řešící problém CPF, jednotlivé přístupy porovnám a ukážu jejich praktické využití. Cílem práce není podat vyčerpávající informace o všech existujících algoritmech, práce prezentuje výběr z nejpoužívanějších algoritmů.

2 Problém Cooperative Pathfinding

Problém cooperative pathfinding (CPF) se zabývá nalezením takové dočasné cesty, kde si agenti navzájem neblokují cestu (nevznikají kolize). Agenti se potřebují přemístit z nějakého výchozího bodu do předem určeného bodu cílového. Všeobecně přijatá myšlenka abstrakce tohoto problému je taková, že stavový prostor si můžeme představit jako neorientovaný graf s agenty umístěnými ve vrcholech grafu tak, že ve vrcholu se může nacházet nejvýše jeden agent a minimálně jeden vrchol zůstane neobsazen. Pohyb je umožněn pouze po hranách grafu do vrcholu, který není obsazen jiným agentem.

Na zobrazení stavového prostoru může být využit libovolný **neorientovaný graf**, po kterém se agenti pohybují. Nechť $G = (V, E)$ je graf, kde $V = \{v_1, v_2, \ldots, v_n\}$ je konečná množina vrcholů a $E \subseteq \binom{V}{2}$ je množina hran. Umístění agentů v grafu odpovídá přiřazení agenta k vrcholu grafu. Nechť $A = \{a_1, a_2, \ldots, a_\mu\}$ je konečná množina agentů, potom rozmístění agentů je

popsáno přiřazovací funkcí (*location function*) $\alpha : A \rightarrow V$.[1]

Problém cooperative pathfinding definujeme jako čtveřici $\sum = [(V, E), A, \alpha_0, \alpha_+]$, kde přiřazovací funkce α_0 a α_+ definují počáteční a koncové rozestavení agentů A v grafu G.

3 Praktické využití

Problém cooperative pathfinding je široce používaný. Je důležitý nejen pro robotiku jako například pro pohyb robotické ruky, rovněž v běžném životě se s ním setkává každý z nás. Například ve špičce, kdy se všichni chtějí dostat třeba z práce domů, jsou silnice přeplněné auty, viz obrázek 2. Musíme tedy jednat tak, aby se jednotlivá auta nesrážela. Kdyby si každý jel tak, jak ho jen napadne, pojišťovny by zkrachovaly. Musí tedy fungovat nějaký určitý řád. Další podobný příklad využití je plánování jízdního řádu vlaků. Vlaky mohou jet po stejné koleji, pokud jedou stejným směrem. V opačném případě by to znamenalo kolizi. CPF problém je rovněž brán v potaz v digitálním průmyslu, viz počítačové hry typu Starcraft a podobné.

Obrázek 1: (ne)kolize v dopravě

[1]Interpretace přiřazovací funkce je taková, že $a \in A$ je umístěno ve vrcholu $\alpha(a)$.

4 Algoritmy řešící problém CPF

Problémem cooperative pathfinding se lidé zabývají již delší dobu. Hledá se co nejjednodušší a nejefektivnější způsob, jak hledání cesty v multi-agentním prostředí zautomatizovat. Klasický algoritmus A* (zde o něm dále nebude řeč) na problémy o více agentech nestačí, proto se začaly vyvíjet nové algoritmy. Jejich stručný popis najdete dále.

4.1 Local Repair A*

Local Repair A* algoritmus (dále jen LRA*) popisuje rodinu algoritmů bohatě zastoupenou v počítačovém průmyslu, hlavně ve videohrách. Každý agent hledá cestu do cíle použitím A* algoritmu, ignoruje všechny ostatní agenty kromě těch nejbližších sousedních agentů. V tu chvíli agenti vyrazí každý svojí cestou, dokud nehrozí kolize. Jakmile je kolize nevyhnutelná (agent by se musel posunout na místo, které je zabrané jiným agentem), okamžitě si přepočítá jinou cestu. Základní algoritmus opovídá prohledávání hrubou silou.

Při použití tohoto algoritmu jsou smyčky možné, ba dokonce i běžné, proto je tedy běžné zkoušet přidat nějaké modifikace, abychom se takovým problémům vyhnuli. Jedna možnost, kterou použijeme zde v tomto článku, je zvýšení aktivity (*agitation level*) pokaždé, když je nucen změnit svou trasu. Náhodný šum je poté přidán do heuristiky vzdálenostiv poměru ke zvýšení aktivity. Jelikož se agenti pohybují stále více náhodně, předpokládá se, že se dostanou mimo problematické místo a zkusí jít jinou cestou.

Je známo, že LRA* má několik závažných nedostatků při obtížných prostředích. Jeho slabiny se projeví v místech, na kterých je v jednom čase více agentů. Řešení v takovém případě může trvat velmi dlouho. Agenti se stále snaží přepočítávat cestu ve snaze uniknout, což ale vyžaduje přepočítávání celého A* algoritmu při každém tahu. To vede k nežádoucímu chování, které je označováno za „neinteligentní“. Každá změna v cestě je plánována nezávisle v kruzích, takže ty stejné pozice mohou být navštíveny opakovaně.

4.2 Cooperative A*

Cooperative A* (CA*) je nový algoritmus pro řešení problému cooperative pathfinding. Úkol je rozdělen do sérií prohledávání pro jednotlivé agenty. Individuální prohledávání jsou realizována v trojrozměrném časoprostoru, při-

čemž počítají s plánováním cest pro jiné agenty. Aby bylo umožněno agentovi zůstat nehybným, do množiny možných akcí agentů je přidán i stav „nulový pohyb“ (*wait move*).

Tabulka rezervací[2] reprezentuje agentovy sdílené informace o plánovaných akcích všech ostatních agentů. Všechny položky zanesené v tabulce rezervací jsou považovány za „neprůchozí“ – žádný agent se nesmí na ten konkrétní uzel přesunout –a hledání cest dalších agentů se těmto uzlům vyhýbá.

Jednoduchá implementace, kterou zde používáme, je založena na tom, že zacházíme s tabulkou rezervací jako s trojrozměrnou mřížkou (dva prostorové rozměry a jeden časový). Každé políčko mřížky, které se nachází v plánované cestě konkrétního agenta, je označeno jako neprůchozí (*impassable*) přesně po dobu trvání pobytu agenta na onom políčku, což zajišťuje, že jiní agenti si nenaplánují svoji trasu, při které by mohlo dojít ke kolizi. Jen malý poměr políček mřížky pozic je takto zabráno, proto tato mřížka může být jednoduše implementována jako hashovací tabulka (*hash table*), hashování probíhá na náhodné distribuční funkci s klíčem (x, y, t).

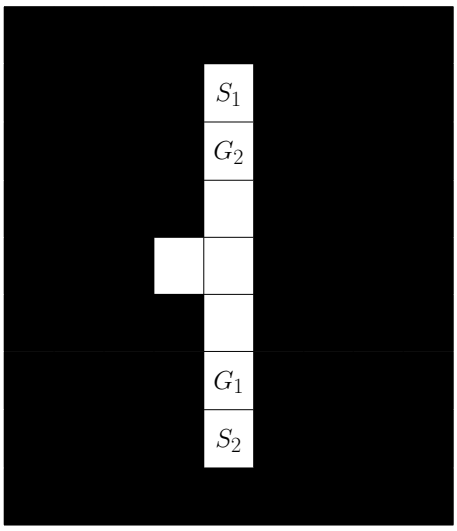
Je důležité poznamenat, že jakýkoli rozdělený hladový algoritmus, který přepočítává optimální cestu, nebude schopen vyřešit jisté třídy problémů. To může nastat tehdy, když hladové řešení pro jednoho agenta brání nějakému řešení pro jiného agenta. Například viz tabulka 1. Obecně jsou tyto algoritmy senzitivní na pořadí agentů. Aby algoritmus fungoval co nejlépe, je důležité dobře agentům připřadit jejich priority, např. lze použít Latombovo prioritizované plánování.

Jakákoli přípustná heuristika může být využita v CA*. Často se používá manhattanská vzdálenost i přes to, že může dávat nevyhovující výsledky ve složitějších prostředích. Ideálně by se měly používat lepší heuristiky, aby se pomohl zredukovat výpočet.

4.3 Hierarchical Cooperative A*

Jsou dvě obecné metody, které pomohou zlepšit heuristiku založenou na abstrakci stavového prostoru. První přístup je předpočítat všechny vzdálenosti v abstraktním prostoru a uložit je do databáze vzorů. Nicméně tohle není

[2]Jakmile si agent jednou vybere cestu, musí být zajištěno, že jiní agenti se budou ve svých cestách vyhýbat těmto uzlům, na kterých v daný okažik agent stojí. Toho je dosaženo právě zanesením každého uzlu do tabulky rezervací. Je to přímá datová struktura obsahující záznam o každém uzlu časoprostorové mapy. Každý záznam specifikuje, jestli je daný uzel obsazen, nebo je volný.



Tabulka 1: Tento jednoduchý problém nemůže být řešen CA* algoritmem. Jeden agent musí projít přes S_1 do G_1, zatímco druhý se snaží dostat z S_2 do G_2.

proveditelné v CPF, protože mapování je dynamické a cíl se může kdykoli měnit. Druhá obecná metoda je použít Hierarchical Cooperative A* (dále jen HCA*). S tímto přístupem jsou abstraktní vzdálenosti počítány na požádání, což je v dynamickém kontextu výhodnější. Hierarchie v tomto případě ukazuje na sérii abstrakcí stavového prostoru, každá obecnější než předchozí, a není omezená na prostorovou hierarchii. R. C. Holte poznamenal, že výběr hierarchie je velmi důležitý (kritický). Velké hierarchie mohou dokonce způsobit horší výsledky než malé jednoduché hierarchie.

HCA* používá jednoduchou hierarchii obsahující abstrakci na jedné doméně, která ignoruje jak časovou dimenzi, tak i tabulku rezervací. Jinými slovy, abstrakce je jednoduchá dvourozměrná mapa s odstraněnými agenty. Abstraktní vzdálenosti tak mohou být viděny jako dokonalé odhady vzdálenosti od cílového bodu za předpokadu, že ignorujeme interakci s ostatními agenty. Toto je přípustná a konzistentní heuristika. Navíc nepřesnost heuristiky je determinována pouze obtížností interakce s ostatními agenty (jak moc se musí agent odchýlit od přímé cesty, aby se vyhnul ostatním agentům).

Jedna z otázek HCA* je, jak nejlépe znovupoužít data v abstraktní doméně. R. C. Holte ve své práci představuje tři různé techniky, jak znovu používat již nalezená data. Čtvrtá technika je představena v Silverově stati, která

znamená použít *Reverse Resumable A** (RRA*) prohledávání na abstraktní doméně.

RRA* algoritmus provádí modifikované A* hledání v opačném směru. Hledání začíná v agentově cílovém bodě G a míří do agentova počátečního stavu O. Místo toho, aby v počátečním bodě skončil, prohledávání pokračuje až do té doby, dokud není expandovaný určitý uzel N. Dobře známý fakt o A* s konzistentní heuristikou je, že optimální vzdálenost z počátečního uzlu do uzlu N je známá od té doby, kdy je uzel expandovaný. Když se prohledávání provádí v opačném směru, znamená to, že optimální vzdálenost z N do G je známá, jakmile skončí prohledávání RRA* algoritmu. Jako heuristika je v RRA* použita manhattanská vzdálenost, přičemž je zachována konzistentnost.

Na rozdíl od předchozích prací zabývajících se tímto oborem je RRA* použito pro výpočet abstraktní vzdálenosti na požádání. Kdykoli je požadována abstraktní vzdálenost z N do G, RRA* zjistí, zda N existuje v uzavřené tabulce. Pokud ano, optimální vzdálenost do tohoto bodu je již známa a může být ihned navrácena. Pokud ne, RRA* prohledávání pokračuje, dokud uzel N není expandovaný. Princip výpočtu je vidět na přiloženém pseudokódu 2.

HCA* je přesně jako CA* algoritmus s více sofistikovanou heuristikou a navíc používá RRA* na vypočítání abstraktní vzdálenosti, když je požadována. Pokud žádný agent nebrání v nejkratší cestě do cílového uzlu, potom první volání do RRA* by mělo obsahovat vzdálenosti do všech požadovaných uzlů. Toho je dosaženo převrácením pořadí funkce následovníků v RRA*, aby bylo zajištěno, že vazby jsou rozbité ve stejném směru (tedy pravá větev v dopředném směru by se měla stát levou větví ve směru opačném).

Pokud jiní agenti stojí v cestě do cílového bodu, HCA* vytlačí všechny nejkratší cesty. V takovém případě cesta povede i přes jiné uzly a RRA* prohledávání musí pokračovat, dokud se nedosáhne vytyčeného cíle. Při každém obnovení RRA* expanduje uzly v soustředných ekvidistantních kruzích od nejkratší cesty až do té doby, dokud není nalezen požadovaný uzel.

4.4 Windowed Hierarchical Cooperative A*

Jednou otázkou u předchozích algoritmů je, jak skončí, když agenti dosáhnou jejich koncového bodu. Pokud se agent nachází v koncovém bodě, na příklad v úzké chodbě, může tak blokovat části mapy ostatním agentům. Ideálně by agenti měli pokračovat ve spolupráci, dokud všichni nedosáhnou kýžené destinace, takže agent se musí posunout i ze svého cílového uzlu, aby tak

```
1: procedure INITIALISERRA*(O, G)
2:     G.g ← 0
3:     G.h ← MANHATTAN(G, O)
4:     Open ← {G}
5:     Closed ← ∅
6:     RESUMERRA*(O)
7: end procedure

8: procedure RESUMERRA*(N)
9:     while Open ≠ ∅ do
10:        P ← pop(Open)
11:        Closed ←(add)— P
12:        if P = N then
13:            return success
14:        end if
15:        for all Q ∈ reverse(SUCCESSORS(P)) do
16:            Q.g ← P.g + COST(P, Q)
17:            Q.h ← MANHATTAN(Q, O)
18:            if Q ∉ Open and Q ∉ Closed then
19:                Open ←(add)— Q
20:            end if
21:            if Q ∈ Open and f(Q) < f(Q in Open)
                   then
22:                Open ←(update)— Q
23:            end if
24:        end for
25:    end while
26:    return failure
27: end procedure

28: procedure ABSTRACTDIST(N, G)
29:    if N ∈ Closed then
30:        return g(N)
31:    end if
32:    if RESUMERRA*(N) = success then
33:        return g(N)
34:    end if
35:    return +∞
36: end procedure
```


Obrázek 2: pseudokód HCA*

umožnil průchod ostatním agentům.

Druhou otázkou je citlivost na pořadí agentů. Ačkoli je občas možné obecně přiřazovat agentům jistou prioritu a některým tak dát prioritu vyšší než jiným, mnohem robustnější řešení je takové, kdy dynamicky měníme pořadí agentů, takže každý agent bude mít po určitou krátkou dobu tu nejvyšší prioritu. Takto mohou být nalezena řešení, což by při předem daném, fixním pořadí nemuselo být možné.

Třetí otázkou je to, že předchozí algoritmy musely počítat kompletní cesty do cílového uzlu ve velkém, třídimenzionálním stavovém prostoru. Při prohledávání cesty u pouze jednoho agenta je plánování a plán realizace prokládány, aby se tak dosáhlo větší efektivity tím, že se vyhneme potřebě plánovat všechny možné následky, které se nám ve výsledku ani nenaskytnou. WHCA* právě nabízí podobnou myšlenku pro CPF.

Jednoduché řešení všech těchto problémů je závádění „oken“ (*window*). Prohledávání při spolupráci agentů (cooperative search) je limitováno na fixní hloubku specifikovanou konkrétním oknem. Každý agent hledá parciální cestu do jeho cílového uzlu, následně se po ní vydá. V pravidelných intervalech (např. když je agent na půl cesty v jeho parciální cestě) se okno posunuje dopředu a vypočítá se nová parciální cesta.

Abychom se ujistili, že agent jde správným směrem, jen hloubka kooperativního prohledávání je limitovaná na fixní hloubku, zatímco abstraktní prohledávání je provedeno do plné hloubky. Okno o velikosti w může být viděno jako střední abstrakce, která je ekvivalentní základní úrovni (base level) stavového prostoru pro w kroků, a ekvivalentní abstraktní úrovni stavového prostoru pro zbytek prohledávání. Jinými slovy, ostatní agenti jsou bráni v potaz pro w kroků (přes tabulku rezervací) a jsou ignorováni po zbytek prohledávání.

Aby se toto prohledávání stalo efektivním, může se použít jednoduchý trik. Když uplyne w kroků, agenti jsou ignorováni a prohledávací prostor se stane identický s abstraktním prohledávacím prostorem. To znamená, že abstraktní vzdálenost poskytuje stejnou informaci jako dokončení prohledávání. Pro každý uzel N_i dosažený po w krocích je určena jedna hrana vedoucí přímo z uzlu N_i do cílového uzlu G o stejné ceně jako je abstraktní vzdálenost z uzlu N_i do cílového uzlu G. Použijeme-li tento trik, prohledávání je zredukováno na okno o w krocích a následně se použije heuristika vysvětlená u HCA*.

Navíc, prohledávání za použití okna může pokračovat, i když agent už dosáhne svého cílového uzlu. Agentovým cílem již není dosáhnout cílového

uzlu, ale dokončit okno přes koncovou hranu. Jakákoli sekvence w kroků tedy povede k dosažení cíle. Nicméně, WHCA* prohledávání efektivně najde sekvenci o nejmenší ceně. Tato optimální sekvence s nejnižší cenou reprezentuje parciální cestu, která dovede agenta co nejblíže k jeho cílovému uzlu a nechá ho tam stát po co nejdelší dobu to bude možné.

Obecně, funkce ceny hran pro WHCA* je:

$$\text{Cost}(P,Q) = \begin{cases} 0 & \text{pokud } P = Q = G, t < w \\ \text{AbstractDist(P,G)} & \text{pokud } t = w \\ 1 & \text{jinak} \end{cases}$$

Další výhodou používání oken je to, že čas zpracování může být rozložen mezi všechny agenty. Máme-li n agentů a okno o velikosti w, pro přepočítávání vzdáleností ve středu každého okna potřebujeme pouze $2n/w$ prohledávání v jednom kole. Pokud se jedno kolo skládá z hodně rámců, v takovém případě se potlačitelné hledání přirozeně rozpadne a dále je rozděleno do různých rámců.

Konečně, výsledky RRA* prohledávání mohou být znovu použity pro každé další okno. Toto ale vyžaduje, aby si každý agent ukládal Open a Close list. Pro každého agenta se provede základní RRA* prohledávání z původní pozice O do jeho cíle G. Pro každé další okno je prováděno další prohledávání, aby se započítaly všechny další uzly. Aby byla zachována konzistentnost, prohledávání musí pokračovat k agentově původní pozici O, ne k aktuální. To znamená, že expanze budou prováděny v soustředných kruzích o stejných poloměrech nad originální nejkratší cestou. Účinnost tohoto přístupu se zmenší, pokud budou agenti nuceni sejít z originální cesty. Nicméně to, co ušetříme při znovuprohledávání, by mělo převážit jakékoli další expanze způsobené odchýlením se od původní cesty.

Demo algoritmu WHCA* lze najít na url http://www.jkilavuz.com/whca/.

5 Závěr

Problém cooperative pathfinding se dá řešit technikami, která více či méně sofistikovaně koordinují pohyb více agentů. Pokud je v prostředí více agentů, bývá výhodné, aby mezi sebou agenti uměli komunikovat ohledně svých cest. Jestliže plánují dopředu v čase stejně jako v prostoru, mohou se agenti vzájemně vyhnout a předejít tak kolizím.

Přehled algoritmů uvedených v této práci rozhodně není kompletní, zajišťuje pouhý vhled do problematiky CPF a udává směr, kterým se lze při studiu problému cooperative pathfinding ubírat.

Reference

[1] Surynek, Pavel: *A SAT-Based Approach to Cooperative Path-Finding Using All-Different Constraints*. Proceedings of SOCS 2012.

[2] Silver, David: *Cooperative Pathfinding*. In Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, červen 1-5, 2005, Marina del Rey, California, USA., nakladatelství AAAI Press. ISBN 1-57735-235-1.

[3] Latombe, J.-C.: *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, USA, 1991, 651 s.