Hierarchical– and Partial–Refinement Pathfinding

Lukáš Bajer

Matematicko-fyzikální fakulta, Univerzita Karlova, Praha
bajeluk (zavinac) matfyz (tecka) cz

Abstrakt Dokument uvádí a snaží se porovnat dva rozdílné přístupy k hierarchickému hledání cest. Prvním z nich je algoritmus HPA* (Hierarchical Pathfinding A*), druhým pak PRA* (Partial–Refinement A*). Oba si kladou za cíl hledat "téměř" optimální cesty na dlažicových mapách s "výrazně" menšími časovými nároky než klasické algoritmy jako A* nebo IDA*

1 Úvod

V prostředí real-timových počítačových her, zvláště pak real-timových strategií, je velmi často potřeba hledat nejkratší cesty po mapě, a to často ještě mnoha agenty nebo jednotkami najednou. Vzhledem k výkonům dnešních počítačů je nutné, aby prohledávací algoritmy pracovaly opravdu rychle, nebo aby své výpočty dokázaly nějakým způsobem rozprostřít v čase. V opačném případě se hra při prohledávání může zpomalovat nebo zasekávat, což výrazně zhoršuje zážitek z hraní a tedy v důsledku i komerční úspěch dané hry.

Aby dnešní standard algoritmů na prohledávání stavového prostoru, algoritmus A*, ve své klasické podobě našel s jistotou cestu mezi startovní a cílovou pozicí, musí při hledání zpracovat alespoň všechny dlaždice na dané cestě, spíše však (vlivem překážek) ještě o poznání více. To může vyžadovat příliš mnoho času a paměti. Vznikly sice mnohé zrychlení použitím lepších datových struktur nebo ořezáváním prohledávaného stromu (viz např. [Astar]), hierarchické algoritmy jdou však podle mě více k podstatě věci a podle praktických testů evidentně znamenají výrazné zlepšení.

1.1 Vztahová poznámka

Na téma hledání cest jsem narazil už v prvním ročníku při programování zápočtového programu. V té době jsem nevěděl nic o tom, že existuje něco jiného než Dijkstrův algoritmus, a tak jsem úlohy, které byly potřeba, řešil tak, jak mě napadlo. Stejně mě ale daný styl programování zaujal a tak jsem se rozhodl v tomto tématu pokračovat i v bakalářské práci. Algoritmy PRA* a HPA* a jiné související články mi rozšířily pohled na danou problematiku, což se jistě bude hodit.

2 Základní filozofie

Jak již bylo naznačeno, oba algoritmy se snaží využít k hledání cest jakési hierarchie nebo prohledávání v několika úrovních, chcete-li. Oba nejdříve nad daným prosorem/mapou vybudují svoje struktury, kterých pak využívají k rychlejšímu hledání cest, ale každý na to jde přeci jenom trochu jinak.

2.1 Hierarchical Path-finding A*

Algoritmus si v iniciační fázi rozdělí danou mapu na tzv. *klastry*, což jsou pravidelné čtvercové oblasti. Praktické testy ukázaly, že vhodnou velikostí pro standardní mapy je 10×10 políček. Vybraná políčka, která bezprostředně sousedí s volnými políčky sousedních klastrů, pak určí za tzv. *průchody*. Tyto průchody se spojí jednak se sousedními průchody z jiných klustrů, jednak navzájem v rámci jednoho klastru, pokud mezi nimi existuje cesta. Délky cest v rámci jednoho klustru se ukládají. Tímto způsobem se vytvoří tzv. *abstraktní graf*.

Postup vytváření klastrů a abstraktního grafu lze iterovat a vytvářet tak zmiňovanou hierarchii. Lze tedy "poodstoupit" výš a budovat nad dosavadními klastry klastry větší (cca dvounásobné) velikosti. Tím se zmenší velikost abstraktního grafu a zrychlí se v něm hledání cest.

Při hledání určité cesty se s rostoucími klastry stává nářočnější operace začlenění startovního a cílového políčka dané cesty do abstraktního grafu, zvětšování klastrů tedy není vhodné dělat do nekonečna (pro mapy velikosti cca 100×100 jsou vhodné 2–3 vrstvy hierarchie). Vlastní nalezení cesty v takto doplněném abstraktním grafu je pak naopak jednoduché a rychlé, použije se standardního algoritmu A* na graf, který má cca setinový počet vrcholů oproti původní mapě.

Abychom získali seznam políček, kudy skutečně povede cesta po mapě, je nutné provést tzv. *path refinement*, neboli zjemnění cesty. Tato operace se s výhodou může provádět postupně až když je potřeba konkrétní úsek cesty. Při programování umělých bytostí se totiž může snadno stát, že se řízení předá jiné části programu, nebo že naplánovaná cesta selže, takže počítání celé cesty dopředu může být často zbytečné.

2.2 Partial-Refinement A*

Podobně jako v předchozím případě, i tento algoritmus používá hierarchii abstraktních grafů, její struktura i práce s ní je ale odlišná. Algoritmus se už na základní mapu dívá jako na graf: vrcholy odpovídají dlaždicím a hrana mezi nimi vede v případě, že spolu odpovídající dlaždice sousedí. Berou se při tom v potaz kromě čtyř základních směrů i čtyři směry diagonální. Místo překrytí mapy strukturou shora (jako to dělá HPA*) PRA* slučuje vrcholy na základě lokálních vlastností grafu.

Algoritmus hledá ve zpracovávaném grafu *kliky*[1], které nahrazuje v budované (vyšší) vrstvě jedinými stavy. Spojováním klik je zaručeno, že všechny původní

[1] *klika* je úplný indukovaný podgraf

vrcholy náležící k novému stavu jsou od sebe vzdáleny právě jeden krok. Tedy skoro: ke klikám se do jednoho stavu totiž přidávají také přilehlí *sirotci*, což jsou vrcholy připojené k dané klice pouze/právě jednou hranou. Celý algoritmus zpracování grafu je určen k iteraci, dokonce ještě více než u HPA* – abstrakce končí v momentě, když už zbyde jenom jeden vrchol. Celá sada vrstev tvoří jakýsi strom, který PRA* využívá.

Partial-Refinement A* pro počáteční (*start*) a koncový (*goal*) vrchol nejdříve zkonstruuje takovou část stromu, aby *start* a *goal* byli v listech, a aby měli jediného společného předka. Poté určí hladinu, od které začne iterovaně vylepšovat cestu – nesmí být ani moc hluboko (algoritmus by provedl příliš mnoho kroků) ani moc mělko (výsledná cesta by se mohla od optimální hodně lišit). V každé iteraci vezme v aktuální hladině cestu z vrcholu "obsahující" *start* do vrcholu, který vznikl abstrakcí vrcholu *goal*, rozbalí jednu úroveň níž a v tomto "pásu" najde nejkratší cestu pomocí A*.

Také PRA* umožňuje vylepšovat cestu *postupně* (tj. jaksi cestou), dokonce ho lze přímo parametrizovat, kolik dlaždic dopředu má počítat. Je evidentní, že je na tuto vlastnost připraven podstatně lépe než HPA*.

3 Porovnání algoritmů

Rychlost a paměťová náročnost. Přesné srovnání nemám k dispozici. Je to tím, že [Hiear] se soustřeďuje spíše na paměťovou složitost, zatímco [PartRef] se více věnuje porovnání rychlosti vůči obyčejnému A*. Přesto lze říci, že oba algoritmy spotřebují až desetkrát méně výpočetního výkonu CPU, což je výrazná úspora. HPA* spotřebovává na svůj OpenList přibližně třetinu oproti běžnému A*, počet navštívených dlaždic je až desetkrát menší.

Přesnost. HPA* produkuje o cca 6 % delší cestu než A*, když se však zařadí postprocessing (smoothing), jeho chyba je kolem 0.5 %. Pokud PRA* počítá celou cestu naráz, i jeho chyba je kolem 0.5 %, pokud cestu zpřesňuje cestou, pohybuje se chyba kolem 5 %.

Akceptovaný typ mapy. Oba algoritmy jsou v originálu implementovány s osmisměrnými dlaždicemi. U HPA* bych neviděl důvod, proč by měl fungovat jinak se čtyřšměrnými, PRA* se však zřejmě zhorší, protože namísto klik o čtyřech vrcholech budou největší kliky pouze dvouvrcholové.

Zdá se, že HPA* vyžaduje pro svou práci v podstatě souvislou 2D mapu, naproti tomu PRA* by zřejmě dobře fungovalo i např. v projektu Enti, kde jsou diskrétní místnosti a celková mapa není (alespoň jednoduše) k dispozici. HPA* pokrývá svými klastry celý dostupný prostor, PRA* naproti tomu svoje abstraktní struktury vytváří jenom z přístupných dlaždic – pokud by byly rozsáhlé plochy na mapě nepřístupné, HPA* by bylo mírně neefektivní.

Implementace. Můj subjektivní názor je, že jednodušší jak na pochopení, tak na implementaci, je HPA*. Článek [PartRef] byl uveřejněn v dubnu tohoto roku a

je z něj patrné, že se algoritmus stále vyvíjí a pracuje se na něm, HPA* vypadá oproti tomu dotaženěji.

4 Celkový dojem

Přečtení a pochopení [Hiear] mi v podstatě nedělalo žádný problém, článek je napsán poměrně populárně, v příloze jsou vysvětlující obrázky a tak první dojem na mě udělal rozhodně lepší HPA*.

[PartRef] byl přesný opak – článek je to poměrně náročný a dalo mi práci jisté věci vztřebat. Zdá se však, že PRA* je obecnější. Přijde mi, že je lépe připraven na zakomponování k dalším činnostem, je více přizpůsoben použití s reaktivním plánováním, potažmo v umělých bytostech. Nakonec si mě získal více.

Reference

[Hiear] Botea, A., Müller, M., Schaeffer, J.: Near Optimal Hierarchical Path-Finding. Dep. of Computer Science, University of Alberta (2004)

[PartRef] Sturtevant, N., Buro, M.: Partial Pathfinding Using Map Abstraction and Refinement. Amer. Assoc. for AI (www.aaai.org) (2005)

[Astar] Higgins, D. F.: Pathfindig Design Architecture, How to Achieve Lightning-Fast A*, AI Game Programming Wisdom. Charles River Media (2002)