Samenvatting Heuristic Search Strategies

Robin Kruit - 0936014

15 mei 2019

1 Introductie

Dit artikel toont de voordelen van een "informed search" strategie vergeleken met een "uninformed search". De algemene aanpak gebruikt een variant van de generieke boom- en graafzoektechnieken genaamd "best-first search". Deze techniek gebruikt een evaluatiefunctie om de volgende node te kiezen. De meeste van deze evaluatiefuncties gebruiken een heuristische functie om te schatten wat het goedkoopste pad is van de staat in node n naar de doelstaat.

2 Greedy best-first search

Deze techniek probeert de node die het dichtste bij het doel is eerst uit te breiden. Zijn evaluatiefunctie zal dus puur uit de heuristische functie bestaan. Deze techniek heeft als voordeel dat de kosten van het zoeken minimaal zijn. Hij is echter niet optimaal en incompleet. De worst-case tijd en ruimtecomplexiteit zijn beide $O(b^m)$.

3 A*

De meest bekende vorm van "best-first search" heet "A* search" (A-star). Het combineert de kosten van het bereiken van een node met de kosten om van de node naar het doel te komen. De goedkoopste oplossing kan gevonden worden door de node met de laagste combinatie van deze twee waardes te zoeken. Indien de tweede waarde, oftewel de heuristische functie, voldoet aan twee condities is A* compleet en optimaal. De tweede conditie geldt enkel als A* gebruikt wordt voor het zoeken in een graaf. Deze twee condities zijn:

- De functie moet een "admissible heuristic" zijn, de functie mag *nooit* de kosten van het behalen van het doel **over**schatten.
- De functie moet consistent zijn, dat wil zeggen de kosten om het doel te behalen vanaf een node mag *nooit* groter zijn dan kosten om de node te bereiken vanaf zijn voorganger plus de kosten om het doel te behalen vanaf de node.
  $h(n) \leq c(n, a, n') + h(n')$

Alle consistente heuristische functies zijn "admissible". Een tree doorzoeken met A* is dus optimaal als de heuristische functie "admissible" is en een graaf doorzoeken is optimaal als de functie consistent is. A* is ook optimaal efficiënt onder optimale algoritmes van dit type. Het nadeel van A* is dat de hoeveelheid staten exponentieel is in de lengte van de oplossing. Omdat A* alle nodes in geheugen houdt is de kans ook zeer groot dat er geheugentekort plaats zal vinden. Hier zijn echter ook oplossingen voor bedacht.

4 Memory-bounded Heuristic search

De simpelste manier om minder geheugen te gebruiken is het implementeren van "iterative deepening". Het algoritme dat hieruit volgt wordt iterative-deepening A* genoemd (IDA*). Het verschil tussen gewone "iterative deepening" algoritmes en IDA* is dat in plaats van de diepte hier de kosten die volgen uit de evaluatie functie gebruikt worden als afkappunt. Het artikel noemt als nadeel van dit algoritme dat het dezelfde problemen heeft met "real-valued costs" als de iteratieve versie van "uniform-cost search".

Twee andere algoritmes zijn RBFS en SMA*. "Recursive best-first search" probeert de standaard "best-first search" na te doen, maar met linear geheugen. Het houdt de evaluatiefunctie waarde van het beste alternatief bij. Op deze manier kan het, als het het huidige limiet bereikt heeft, terugvallen op dit alternatief. Dit algoritme is efficiënter dan IDA*, maar leidt onder dezelfde soort problemen.

"Simplified memory-bounded A*" stelt een limiet aan het hoeveelheid geheugen dat gebruikt mag worden. De slechste nodes worden opgeruimd voor nieuwe nodes indien dit geheugen geheel gebruikt is. Dit algoritme is compleet als er een bereikbare oplossing is en optimaal als een optimale oplossing bereikbaar is. Bij zeer complexe problemen zal het wisselen tussen mogelijke paden in het geheugen echter leiden tot tijdsverlies.

5 Learning to search better

Algoritmes kunnen leren beter te zoeken door een "metalevel state space" bij te houden. Door bij te houden welke stappen vaak niet leiden tot succesvolle oplossingen en deze te vermijden kunnen algoritmes proberen de totale zoekkosten te minimaliseren.

6 The effect of heuristic accuracy on performance

De kwaliteit van de gebruikte heuristische functie kan een groot verschil maken. Een manier om deze kwaliteit te meten is het "effective branching factor" ($N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$). Deze factor is redelijk consistent voor moeilijke problemen, het experimenteel meten van deze b* kan de effectiviteit van de heuristische functie dus aansturen. Een strengere functie is over het algemeen beter, ervan uitgaande dat deze consistent en niet te complex is.

7 Generating admissible heuristics from relaxed problems

Een probleem met minder restricties op zijn acties is een "relaxed problem". De graaf die dit probleem beschrijft is groter omdat er meer acties mogelijk zijn. Een optimale oplossing in de kleine graaf is per definitie ook een oplossing in de grotere graaf. Een optimale oplossing in de grote graaf kan beter oplossingen hebben. De kosten van deze betere oplossingen zijn dus een "admissible heuristic" voor het gewone probleem en zijn ook consistent. Hierbij is het cruciaal dat deze oplossingen goedkoop te genereren zijn. Indien meerdere oplossingen mogelijke kandidaten zijn kan de hoogste waarde genomen worden.

8 Generating admissible heuristics from subproblems: Pattern databases

"Admissible heuristics" kunnen ook afgeleid worden van de oplossingskosten van een subprobleem. Een "Pattern database" slaat de exacte oplossingskosten van alle subproblemen op. De "admissible heuristic" wordt dan gegeneerd door voor elke complete staat het bijbehorende subprobleem op te zoeken en de oplossingskosten hiervan te gebruiken. De heuristiek van de

subproblemen kunnen helaas niet simpelweg opgeteld worden om tot het gehele probleem te komen omdat er hoogstwaarschijnlijk overlap in de stappen tot de oplossing zal zitten. Wel kan de som van de twee subproblemen gezien worden als een ondergrens voor het oplossen van het gehele probleem. Dit kan leiden tot een gigantische verkleining van de hoeveelheid nodes die gegenereerd moeten worden.

9 Learning heuristics from experience

Een andere manier om een heuristische functie te generen is uit ervaring. Ervaring is hier het vele malen oplossen van soortgelijke problemen. Door de staat van het oplossingspad en de kosten van de oplossing te gebruiken in een leeralgoritme kan een functie gegenereerd worden. Deze functie kan dan proberen de kosten te voorspellen.

Inductieve leermethoden werken door kenmerken van de state te gebruiken in plaats van de state zelf. Vaak worden deze lineair gecombineerd om tot de heuristische functie te komen. Deze functies zijn niet noodzakelijk "admissible" of consistent.