Řešení MAPF Problem pomocí Conflict-based search algoritmu

Matouš Kovář
Fakulta informačních technologií, České vysoké učení technické

April 27, 2024

Úvod

V této práci jsem se zaměřil na řešení MAPF Problem pomocí Conflict-base search(CBS) algoritmu. U MAPF problému řešíme nalezení cest agentů v prostoru tak, aby se v žádném časovém okamžiku agenti nestřetli a minimalizovali cenu cesty. CBS funguje na principu hledání cest pro jednotlivé agenty samostatně a poté hledá kolize v jednotlivých cestách, které se snaží odstranit. Pro zjednodušení problému uvažujeme pouze mřížku o rozměrech N x N bez překážek s libovolným počtem agentů. Toto téma jsem si zvolil jelikož mě samotného zajímalo jak vnitřní inteligence agentů funguje a není to tak často probírané téma jako například systematické prohledávání prostoru pro jednoho agenta.

Algoritmus

Jak již bylo zmíněno, algoritmus hledá cesty pro jednotlivé agenty nezávisle a následně je porovnává mezi sebou. Pokud v cestách dvou agentů nastane jedno z následujících:

- V n-tém kroku se dva agenti nacházejí na stejném místě
- Agenti si prohodí pozici

tak nastává kolize.

Kolize

Pokud v cestách agentů nastává kolize, tak není řešení validní, potřebujeme tedy zavést omezení - Constraint. Pokud se dva agenti nacházejí v n-tém kroku na stejné pozici, tak musíme nalézt cestu takovou, že buď první, nebo druhý agent nepřejde v n-tém kroku kolizní pozici. Jak se toto řeší konkrétné budeme řešit v následujícím odstavci.

Celý proces prohledávání funguje na dvou hladinách:

1. High Level
2. Low Level

Low Level

Tato hladina je zodpovědná za nalezení cesty pro jednoho agenta. Vnitřní implementací bude tedy nějaký algoritmus pro systematické prohledávání prostoru. V našem případě jsem použil algoritmus A*. Algoritmus musí mít ale drobnější modifikaci. Jelikož jsme výše zjistili, že v cestách může docházet ke kolizím, tak přidáme algoritmu jako vstup pole Constraints - které obsahuje omezení pro agenta - na jakých pozicích se nesmí v určitý čas nacházet. V tomto prostoru poté nalezneme novou cestu.

High Level

Druhá vrstva našeho algoritmu se stará o hledání kolizí a tvoření nových omezení - Constraints pro jednotlivé agenty. Zavedeme si nyní takzvaný Constraint Tree(který) v sobě drží informace o omezeních pro jednotlivé agenty. V kořeni nemá žádné omezení. Po vyřešení samostatných cest hledá první kolizi a vytvoří dva potomky - každý dědí všechny omezení z rodiče a navíc přidává jedno omezení pro agenta, který se nachází v kolizi. Jednotlivé vrcholy jsou přidávány do prioritní fronty, kde jsou řazeny pomocí účelové funkce.

Implementace

Nyní trochu přiblížím způsob mojí implementace a technologie, které jsem použil. Jednotlivé pozice jsou implementovány v třídě *Node*, která uchovává souřadnice a časový okamžik. Jako LowLevel jsem zvolil A*, který je implementovaný ve třídě *AStarSolver*. Jelikož pracujeme ve mřížce tak pro měření vzdálenosti používáme Mannhattan distance:

$$d(M, P) = |M_x - P_x| + |M_y - P_y| \tag{1}$$

Jako heuristickou funkci používáme:

$$h(M) = Cost(M) + d(M, end) \tag{2}$$

kde $Cost(M)$ je délka dosavadní nejkratší cesty do M a end jsou souřadnice cílové pozice. *AStarSolver* má zároveň atribut constraints, který v sobě uchovává

jednotlivé omezení pro agenta. Třída *HighLevel* v sobě drží samotný Constraint Tree. Jednotlivé vrcholy v Constraint Tree jsou implementované ve třídě *CTNode*. Běh algoritmu je implementovaný v metodě *HighLevel.run*() K implementaci priority queue používám knihovnu heapq, která má v sobě implementovanou minimovou haldu a pro vizualizaci pygame.

Algorithm 1 Conflict-based search

ROOT.constraints $\leftarrow \emptyset$
ROOT.paths $\leftarrow$ *LowLevel.solve*(*ROOT*)
ROOT.cost $\leftarrow$ *Price*(*ROOT*)
$q \leftarrow$ empty queue
q.push(*ROOT*)

while q is not empty **do**
  CURRENT $\leftarrow$ node with lowest price from q
  if *CURRENT* has no collisions **then**
    return *CURRENT.paths*
  end if

  for all pair of agent paths in *CURRENT.paths* **do**
    if collision appears at $(x, y, time)$ **then**
      $child1, child2 \leftarrow$create child with new constraint
      add $child1, child2$ to q
    end if
  end for

end while
return False

Diskuse

Ačkoliv moje implementace už značnou část problémů dokáže vyřešit, nabízí se několik otázek.

1. Mohou agenti čekat?
2. Je jinný způsob podle čeho řadit CT Node ve frontě?
3. Lze použít i jiný algoritmus než A*

Čekání

V některých algoritmech, např HCA*[1] se osvědčilo přidat agentům možnost čekat na stejném políčku. Ve zdroji[2] konflikty řeší tak, že jednoho agenta nechají zkrátka čekat na stejném políčku. Toto řešení je jistě možné a minimalizuje délku uraženou jedním agentem. Moje řešení, které minimalizuje počet kroků, které udělá agent preferuje možnost, kdy se agenti vyhnou.

Řazení fronty

V mém případě používáme funkci, která se snaží minimalizovat počet kroků každého agenta - jde nám tedy o co nejkratší čas cesty agenta do cíle. Je ale možné také minimalizovat vzdálenost uraženou jednotlivými agenty, nebo metrika Fuel, která se snaží minimalizovat pomyslné spotřebované palivo - move kroky něco stojí, ale wait ne.

Low Level algoritmus

Pro implementaci lze použít prakticky každý algoritmus pro prohledávání prostoru. Já jsem se rozhodl pro A* z důvodu prostorové úspornosti - expanduje méně vrcholů

Závěr

Povedlo se naimplementovat, fungující algoritmus pro řešení MAPF problému. Algoritmus má jisté mezery - mohla by se přidat možnost čekání agentů, přidání překážek do prostředí, umožnit navštívit stejnou pozici znovu v jiném časovém kroku. Algoritmus funguje ale dobře pro prostory, kde se nevyskytuje velké množství kolizí.

References

[1] David Silver. "Cooperative Pathfinding". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 1.1 (Sept. 2021), pp. 117–122. DOI: 10.1609/aiide.v1i1.18726. URL: https://ojs.aaai.org/index.php/AIIDE/article/view/18726.

[2] Guni Sharon et al. "Conflict-based search for optimal multi-agent pathfinding". In: *Artificial Intelligence* 219 (2015), pp. 40–66. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2014.11.006. URL: https://www.sciencedirect.com/science/article/pii/S0004370214001386.