Bozza di tesi

Alessandro Tellarini

Matricola 975490

Capitolo 1

introduzione

L'intelligenza artificiale è una delle tecnologie più discusse degli ultimi anni: innumerevoli articoli di giornale sono stati scritti su questo tema che, nel giro di qualche decennio, è passato dall'essere principalmente un argomento di fantascienza ad affermarsi come una concreta realtà, che si sta facendo largo nella vita di tutti i giorni.

Quando si parla di intelligenza artificiale molte persone pensano immediatamente ai language model, come l'ormai celebre Chat-GPT, strumenti che possono facilmente dare l'illusione di comunicare con un essere senziente, quasi umano, e che quindi corrispondono a ciò che nell’immaginario comune è un’intelligenza artificiale.

Al di là del sentimento popolare, per poter discutere di IA è importante dare una definizione di “intelligenza”, è infatti impensabile fabbricare un’intelligenza artificiale senza avere prima deciso cosa rende una macchina intelligente.

Nel tentativo di dare una definizione soddisfacente sono nate quattro diverse scuole di pensiero: la prima definisce intelligente un agente che pensa come un essere umano, la seconda un agente che pensa in modo razionale, la terza uno che agisce come un essere umano e la quarta un agente che agisce in modo razionale.

Queste teorie si possono dividere in due gruppi: le prime due seguono il paradigma dell’emulazione, per cui l’intelligenza è un fenomeno emergente, che si può osservare in un sistema che sia internamento progettato come una reale entità intelligente, l’essere umano. Se si riuscisse a comprendere come funziona la mente umana, la si potrebbe codificare in un software che ne rispecchi le qualità. Purtroppo per ora il cervello umano è uno dei più grandi misteri dell’universo; perciò, non è una strada realisticamente percorribile.

Le altre due seguono il paradigma della simulazione, per cui non è importante il funzionamento interno di un’entità intelligente, quanto piuttosto il comportamento che ne deriva. Questo approccio è decisamente più percorribile, in quanto permette di costruire un agente la cui “mente” non debba funzionare come la nostra.

Seguendo il paradigma della simulazione, si deve quindi decidere se optare per un agente che agisca come un essere umano oppure che agisca in modo razionale. La prima opzione non è molto allettante, in quanto spesso gli esseri umani non prendono decisioni sagge; la seconda è infatti quella comunemente seguita.

Un agente si dice dunque intelligente se agisce razionalmente, ossia se prende scelte razionali. Se davanti a diverse possibilità esso prende la scelta più razionale, ossia quella che gli porta il maggior beneficio, allora può essere definito intelligente.

Si può comprendere come, definendo l'intelligenza artificiale in questi termini, si possano comprendere in essa molti più strumenti e tecnologie di quelli che vengono solitamente immaginati. Un ramo estremamente importante ed attuale dell'IA è, ad esempio, quello che riguarda agenti autonomi che devono muoversi nello spazio.

Si tratta di un problema non da poco, un agente intelligente che si muove nel mondo fisico deve essere in grado di risolvere diversi problemi: dovrà poter calcolare percorsi ottimali dalla sua posizione a quella di destinazione e dovrà essere capace di evitare ostacoli, pericoli e altri agenti che si muovono nello stesso spazio.

Il presente elaborato verte su questo tema e nello specifico sul problema del *Multi-Agent Pathfinding* (MAPF).

È un problema difficile da risolvere, con svariate applicazioni: la sua risoluzione è necessaria per poter far lavorare nello stesso ambiente robot capaci di muoversi autonomamente, ad esempio quelli presenti nei magazzini di Amazon. Il problema ha però applicazioni anche in svariati altri settori, tra cui ad esempio la gestione di treni e aeroporti, oppure il *pipe routing*, ossia il posizionamento di tubi in ambito industriale, effettuato senza che questi si compenetrino.

La tesi si concentra su un problema strettamente correlato al MAPF, ossia il *Multi-Agent Meeting problem* (MAM). In questo problema ci sono molteplici agenti ma, a differenza del MAPF, essi non devono recarsi in destinazioni prefissate, ma piuttosto convergere in un'unica posizione, idealmente la migliore possibile.

L'obiettivo del lavoro svolto consiste nel generalizzare il MAM, facendo in modo che gli agenti possano incontrarsi non in un'unica posizione ma in un sottoinsieme di nodi del grafo, caratterizzati dal fatto di essere tutti connessi tra loro.

Sono presenti quindi due sfide principali: trovare un sottoinsieme di nodi completamente connesso, valido come insieme di destinazioni per gli agenti, e fare in modo che il processo seguito porti a prendere la scelta migliore, ossia a indirizzare gli agenti alle posizioni che li costringono a fare complessivamente il minimo percorso possibile.

Capitolo 2

contesto e stato dell’arte

MAPF classico

È innanzitutto importante definire in modo chiaro il problema del *Multi-Agent Pathfinding*.

In quello che si può chiamare un problema *classico* di MAPF, si hanno k agenti e un input costituito da una tupla (G, s, t), dove G = (V, E) è un grafo non direzionato, in cui V è l’insieme dei vertici, mentre E quello degli archi; la funzione s : [1, …, k] -> V assegna ad ogni agente una posizione di partenza sul grafo; la funzione t : [1, …, k] -> V assegna invece ad ogni agente una posizione “target”, ossia quella in cui deve dirigersi.

Il tempo viene considerato come discreto e in ogni timestep ogni agente si trova su un nodo del grafo e può effettuare esattamente una azione. Un’azione è definibile come una funzione a: V -> V tale che a(v) = v’, ossia se un agente effettua l’azione a trovandosi nel nodo v del grafo, nel timestep successivo si troverà nel nodo v’. Esistono azioni di *move* in cui l’agente si sposta dal nodo in cui si trova in uno adiacente e quelle di *wait* in cui l’agente resta fermo dove si trova.

Una sequenza di azioni si può considerare un *single-agent plan* per un agente i se l’esecuzione di tali azioni lo portano a trovarsi nel nodo di goal t(i). Una soluzione è costituita da una serie di k *single-agent plan*, uno per ogni agente presente nel problema.

Una soluzione, per essere considerata valida, deve però garantire che non avvengano collisioni tra gli agenti; è perciò fondamentale definire quali situazioni costituiscano una collisione.

Il tipo più comune di conflitto è il *vertex conflict*,che avviene quando, seguendo i piani a loro assegnati, due agenti andrebbero a trovarsi nello stesso nodo del grafo nello stesso timestep. Un altro conflitto molto comune è l’*edge conflict*, che si verifica quando due agenti dovrebbero attraversare lo stesso arco del grafo, nella stessa direzione e nello stesso timestep. Se invece l’arco è lo stesso ma le direzioni sono opposte, si parla di *swapping conflict* e si hanno quindi due agenti che si scambiano di posto. Sebbene esistano ulteriori tipi di conflitto, questi esposti sono quelli che più comunemente vengono interdetti nei problemi di MAPF.

Un altro aspetto del problema che va chiarito è il comportamento degli agenti una volta raggiunte le loro rispettive destinazioni. Il paradigma più comunemente seguito è lo *stay at target*, dove ogni agente, raggiunto il proprio nodo di goal, rimane fermo in quella posizione; il *disappear at target* invece presuppone che gli agenti “spariscano” una volta arrivati al proprio goal, ossia non vengano più considerati nel calcolo dei conflitti.

Avendo definito i tipi di conflitto che sono vietati, è possibile stabilire se una soluzione sia valida oppure no. A questo punto bisogna avere un modo per poter valutare diverse soluzioni e stabilire quali siano migliori di altre e in che misura; è quindi necessario definire delle funzioni obiettivo. Le due più comunemente usate sono *makespan* (MKSP), che considera il numero di timestep necessari poiché tutti gli agenti arrivino ai rispettivi goal, ossia il costo massimo tra i *single-agent plan*, e *sum of costs* (SOC), che calcola il costo di una soluzione come la somma dei costi di tutti i percorsi degli agenti.

Varianti del MAPF

Esistono diverse varianti del problema che rilassano le assunzioni fatte finora.

In alcuni varianti di MAPF non è vera la regola per la quale ogni azione, di *move* o *wait*, ha costo unitario. Il grafo G in cui gli agenti si muovono può essere definito come un grafo pesato, in cui archi differenti possono avere diversi costi di attraversamento. Precedentemente si è definita l’azione di *move* come quella che fa spostare un agente da un nodo del grado ad uno adiacente; la variante del MAPF in *2k-neighbor grids* permette agli agenti di muoversi verso uno dei 2k nodi vicini e i costi degli archi sono calcolati utilizzando la distanza Euclidea. Il problema può essere ulteriormente generalizzato assumendo che ogni nodo in G rappresenti un punto Euclideo (x, y), nel quale ogni agente possa dirigersi.

Un’altra caratteristica del problema che può variare è l’insieme di requisiti necessari affinché una soluzione sia considerata valida. Solitamente viene richiesto solo che non siano presenti conflitti, ma varianti del problema possono introdurre nuove condizioni.

Possono essere aggiunte regole di robustezza, le quali prendono in considerazione l’eventualità di ritardi nell’esecuzione dei *single-agent plan*. Un piano è k-robusto se non sono presenti conflitti tra gli agenti anche introducendo un ritardo di k timestep. Esistono inoltre regole di formazione, le quali richiedono che gli agenti si muovano tenendo in considerazione la posizione degli altri agenti, ad esempio mantenendo una distanza massima l’uno dall’altro. Queste regole aggiuntive possono essere utili per descrivere in modo più accurato situazioni specifiche, per le quali i parametri classici del MAPF non sarebbero sufficienti.

Un’altra assunzione fatta finora è che gli agenti abbiano tutti la stessa forma e dimensione; ciò non è sempre vero, potrebbero infatti esserci agenti più ingombranti che quindi occupano più spazio. In questo caso potrebbero verificarsi conflitti anche tra agenti che non sono sullo stesso nodo o arco, ma che sono comunque troppo vicini tra loro.

Un’altra estensione importante del MAPF è il *permutation-invariant MAPF* (PI-MAPF), noto anche come *anonymous MAPF*, ossia una variante del problema in cui è presente un insieme di nodi di goal, ma essi non sono assegnati singolarmente ad ogni agente, bensì possono essere raggiunti da qualunque di essi, l’importante è che alla fine ogni agente si trovi in uno dei nodi dell’insieme. Una generalizzazione di questa variante del problema è il *colored MAPF*, in cui gli agenti sono divisi in gruppi o squadre e ogni gruppo ha un sottoinsieme di nodi di goal in cui si deve dirigere.

Un caso speciale di PI-MAPF è lo *shared-goal MAPF* (SG-MAPF), nel quale tutte le posizioni di goal coincidono e di conseguenza il problema diventa trovare percorsi privi di conflitti per ogni agente, dalla sua posizione di partenza a quest’unica posizione di goal.

Esistono anche varianti del problema in cui, una volta che un agente ha raggiunto il suo goal, gliene viene assegnato un altro. In contesti di questo tipo possono anche apparire nuovi agenti, con dei goal da raggiungere.

MAM

Il *multiple agents meeting problem* (MAM) è un problema che presenta similarità con il MAPF, ma ha anche sostanziali differenze rispetto ad esso e tutte le sue varianti.

In entrambi i problemi sono presenti k agenti, per i quali bisogna trovare k percorsi che li portino dalle loro rispettive posizioni di partenza a delle posizioni di goal, ma mentre nel MAPF i percorsi devono essere rigorosamente privi di conflitti, tale condizione non sempre è presente nei problemi di MAM. In questi ultimi, inoltre, gli agenti devono convergere in un’unica posizione di goal, similarmente al SG-MAPF; il problema è utile per rappresentare situazioni dove più agenti possono coesistere nella stessa posizione, come ad esempio coordinare l’incontro di più persone in un unico edificio che le può accogliere tutte.

La differenza più importante tra MAM e MAPF è però nell’input del problema: nelle istanze di MAPF, in tutte le sue varianti, le posizioni di goal sono prestabilite, mentre nel problema di MAM l’unica posizione di goal, o di incontro (spesso chiamata M, da *meeting*) non è data, ma deve essere calcolata. Di conseguenza in questo problema si viene ad aggiungere un ulteriore livello di complessità, in quanto non basta trovare i percorsi privi di conflitti per gli agenti, ma è anche necessario trovare una posizione in cui farli incontrare. La scelta del nodo di incontro non è casuale: esso viene determinato minimizzando una funzione obiettivo, sia essa *sum of costs* o *makespan*, in modo da ridurre il più possibile il costo dei cammini degli agenti. Ciò è ragionevole; se si devono far incontrare degli agenti tra loro, siano essi persone o robot, è logico scegliere un punto che sia “comodo” da raggiungere per tutti, per minimizzare lo sforzo medio.

Come accennato nell’introduzione, lo scopo di questo lavoro è generalizzare il MAM.

Si considererà una sua variante, in cui gli agenti non dovranno incontrarsi in un’unica posizione, bensì raggiungere un insieme di posizioni di goal, che non saranno fornite, ma verranno calcolate. L’insieme di posizioni dovrà inoltre essere scelto facendo in modo che esse formino una cricca, ossia siano tutte connesse tra loro relativamente ad un grafo di connettività che viene introdotto nel problema.

Per calcolare i percorsi privi di conflitti verrà utilizzato l’algoritmo CBS (*conflict-based search*), che verrà introdotto a breve. Il contributo principale del lavoro svolto sta nella risoluzione del problema antecedente, ossia la determinazione di un insieme di nodi adatti ad essere scelti come goal per gli agenti.

ALGORITMI

Esistono diversi algoritmi che possono essere utilizzati per risolvere problemi di MAPF e di MAM. Considerato che, una volta determinato un insieme di nodi di goal per gli agenti, è possibili assegnare ogni agente ad uno di essi e risolvere il problema come un’istanza di MAPF, verranno qui esposti i principali algoritmi esistenti per la risoluzione di problemi di MAPF.

Per classificare gli algoritmi si può utilizzare come criterio la loro ottimalità. Esistono infatti algoritmi ottimali, subottimali e subottimali *bounded*. Logicamente, un algoritmo che restituisce sempre la soluzione ottima, quella con il costo minimo, sembra essere la scelta migliore, ma nella pratica questi algoritmi possono talvolta richiedere tempi di esecuzione piuttosto lunghi, tali da rendere conveniente optare per un algoritmo che non garantisca sempre la soluzione migliore, ma che la restituisca in tempi più brevi.

Tra gli algoritmi subottimali ne esistono diversi categorizzabili come *rule-based* o *procedure-based*, dove cioè si risolve il problema utilizzando delle euristiche, le quali non garantiscono di risolvere il problema ottimamente, ma portano statisticamente a buoni risultati. Un esempio è l’algoritmo PUSH, che prima calcola i piani di movimento per i singoli agenti e poi risolve le collisioni facendo in modo che ogni agente possa “spingere” via altri agenti che lo ostacolino. Se ad esempio l’agente A vuole muoversi dalla posizione p1 alla posizione p2, in cui è presente l’agente B, B dovrà spostarsi, muovendosi in un nodo adiacente e facendo passare A. Algoritmi di questo tipo possono essere decisamente efficienti, ma oltre ad essere subottimali sono anche incompleti, non garantiscono infatti di trovare una soluzione anche se essa esiste. Con l’algoritmo PUSH può capitare che si creino situazioni di stallo tra gli agenti, potenzialmente risolvibili.

Algoritmi ottimi possono invece essere basati su A\*, un algoritmo che viene solitamente utilizzato per risolvere in modo ottimo ed efficiente problemi di *search*, dove si ha un unico agente che deve muoversi da una posizione di partenza ad una di goal, evitando ostacoli e scegliendo il percorso migliore. L’algoritmo più semplice per MAPF da esso derivato è *A\*-based search in the joint space*, dove si definisce come stato una tupla contenente la posizione di ogni agente sul grafo e successivamente si esegue una ricerca con A\* su questo nuovo insieme degli stati, considerando come stato di partenza l’insieme delle posizioni di partenza degli agenti e come stato di goal quello in cui ogni agente si trova nella rispettiva posizione di goal. L’algoritmo è ottimo, ma caratterizzato da una pessima scalabilità, in quanto la complessità computazione cresce esponenzialmente all’aumentare degli agenti, rendendo questa procedura inutilizzabile, se non per un esiguo numero di agenti.

CBS

…

Capitolo 3

analisi del software sviluppato

…

Capitolo 4

prove sperimentali

…

Capitolo 5

analisi dei dati raccolti

…

Capitolo 6

conclusioni e possibili sviluppi futuri

…

Riferimenti bibliografici

…