Tesi Triennale

Alessandro Tellarini

Matricola 975490

Capitolo 1

Introduzione

La continua innovazione e lo sviluppo che stanno interessando i settori dell’automazione e dell’intelligenza artificiale, la cui rilevanza cresce ogni anno, di pari passo con la diffusione dei prodotti da essi derivati in ogni ambito della società, rendono la ricerca di algoritmi di *pathfinding* affidabili ed efficienti una sfida attuale e necessaria. In svariati contesti sono presenti agenti autonomi (*ad es. robot*) che devono muoversi in un ambiente, senza l’intervento dell’uomo, ed è compito di tali algoritmi fare in modo che essi possano svolgere le loro funzioni, senza scontrarsi tra di loro o con degli ostacoli, scegliendo percorsi sicuri.

L’obiettivo di questa tesi consiste nella generalizzazione di un problema strettamente legato a questo contesto, il *Multi-Agent Meeting problem* (MAM), nello specifico la variante *collision-free* (CF-MAM), nel quale per un determinato numero di agenti, posizionati in un ambiente in cui possano muoversi, si ha il compito di trovare un punto di incontro e dei percorsi che essi possano seguire per arrivarci, senza collidere tra loro o con elementi statici dell’ambiente.

Il *Multi-Agent Reconnection problem* (MAR), introdotto in questo elaborato, sostituisce ad un unico punto di incontro un insieme di posizioni, una per ogni agente, in cui essi dovranno dirigersi. Tali punti (*che non saranno forniti in input, ma dovranno essere generati*) avranno la caratteristica di essere un sottoinsieme fortemente connesso di nodi del grafo rappresentante l’ambiente, nel quale da ogni posizione che ne faccia parte, un agente possa comunicare con tutti gli altri. Il problema descrive quindi situazioni dove un gruppo di agenti si trova separato e ha la necessità di riunirsi. Ciò potrebbe verificarsi per diverse ragioni, ad esempio in seguito ad un evento imprevisto che ha costretto gli agenti ad allontanarsi gli uni dagli altri, oppure potrebbe semplicemente rappresentare una fase di *regrouping* al termine di attività individuali degli automi.

Trovare un insieme di posizioni valido non sarà tuttavia sufficiente: la scelta delle destinazioni dovrà essere tale da minimizzare, per quanto possibile, il costo dello spostamento degli agenti, calcolato in base alla lunghezza complessiva dei percorsi da essi presi. L’algoritmo sviluppato dovrà quindi non solo essere corretto e completo (*ossia, rispettivamente, restituire solo soluzioni valide e trovare sempre una soluzione qualora essa esista*), ma dovrà anche essere efficiente e soprattutto affidabile, restituendo quindi mediamente soluzioni dall’elevato grado di ottimalità. In questo elaborato si mostrerà come tale risultato è stato raggiunto, analizzando il codice sviluppato e gli esiti degli esperimenti svolti.

La risoluzione di questo problema può avere un impatto significativo, in quanto esso può essere utilizzato per descrivere una svariata gamma di contesti applicativi. Potrebbe, ad esempio, essere utile per guidare il movimento di gruppi di robot utilizzati nella ricerca di superstiti in seguito a crolli di edifici e disastri naturali, se gli automi in questione avessero la necessità di mettersi regolarmente in condizione di comunicare tra loro, per scambiarsi informazioni sul terreno esaminato da ognuno di essi ed accelerare le ricerche. La stessa necessità potrebbe sorgere anche per robot adibiti all’esplorazione, come ad esempio i rover utilizzati nelle missioni spaziali. Oltre ai robot, anche le automobili a guida autonoma potrebbero trarne beneficio, in quanto potrebbero essere create reti di veicoli che, una volta lasciato il proprio passeggero a destinazione, siano in grado di ritornare a dei *hub* dove ricaricarsi, in attesa della prossima corsa. Questi sono solo alcuni esempi di applicazioni reali che potrebbe avere il *Multi-Agent Reconnection problem*.

Organizzazione della tesi

Il presente elaborato è suddiviso in quattro capitoli, ognuno dei quali riguardante un aspetto della tesi. Il Capitolo 1 fornisce un’introduzione al lavoro svolto e agli obiettivi che esso vuole realizzare. Il Capitolo 2 analizza il contesto all’interno del quale esso si inserisce, considerando i problemi più simili a quello affrontato, dai quali esso deriva, e i principali algoritmi che sono stati realizzati per risolverli, in particolare quello che è stato utilizzato anche in questo progetto. Il Capitolo 3 illustra il codice che è stato sviluppato, spiegandone il funzionamento. Il Capitolo 4 raccoglie e analizza i dati ottenuti dagli esperimenti svolti per testare il software sviluppato.

Capitolo 2

Contesto e stato dell’arte

Come già accennato nell’introduzione, il problema MAR nasce come generalizzazione del già esistente MAM, il quale presenta delle forti similarità con un terzo problema, il *Multi-Agent Pathfinding problem* (MAPF). Questo problema ha una grande rilevanza per questa tesi in quanto, come verrà illustrato nel Capitolo 3, per risolvere le istanze di MAR, esse verranno ridotte a istanze di MAPF, per le quali verrà applicato l’algoritmo noto come *Conflict-Based Search* (CBS). Per questo motivo, è innanzitutto di primaria importanza definire in modo rigoroso MAPF.

2.1 *Multi-Agent Pathfinding* (MAPF)

MAPF è un problema nel quale l’obiettivo è pianificare percorsi per molteplici agenti, ognuno con una posizione di partenza ed una di *goal* alla quale deve dirigersi. Il vincolo principale è quello per cui non devono verificarsi collisioni: gli agenti devono poter seguire i percorsi assegnatigli senza collidere tra loro o con ostacoli. Il problema ha un grande numero di applicazioni, tra cui la gestione di veicoli a guida autonoma e di magazzini automatizzati. Di conseguenza, MAPF ha ricevuto una notevole attenzione a livello accademico nel corso degli anni (Felner et al. 2017; Bartak et al. 2018; Li et al. 2019; Zhang et al. 2020, ecc.).

Nella letteratura esistente rigurardante MAPF le caratteristiche e gli obiettivi del problema non sempre sono gli stessi. In questo elaborato viene considerata la versione “classica”, proposta da Stern et al. [[1](#riferimento_1)].

Un’istanza del problema con k agenti prevede un input costituito da una tupla *(G, s, t)*, dove *G = (V, E)* è un grafo non orientato, nel quale *V* è l’insieme dei nodi e *E* quello degli archi; la funzione *s : [1, …, k] → V* associa ad ogni agente un nodo di *start*, mentre *t : [1, …, k] → V* associa ad ogni agente un nodo di *goal*. Si considera il tempo come discreto, ovvero come diviso in *timestep*, in ognuno dei quali ogni agente si troverà in uno specifico nodo del grafo e potrà compiere una determinata azione. Un’azione viene formalizzata come una funzione *a : V → V*, tale che *a(v) = v′*; ciò significa che se un agente si trova nel vertice *v* del grafo e compie l’azione *a*, allora nel *timestep* successivo si troverà nel vertice *v′*. Gli agenti hanno a disposizione due tipi di azione: *wait*, che fa rimanere nel nodo in cui si trova l’agente fino al prossimo *timestep*, e *move*, che fa muovere l’agente dal nodo *v* in cui si trova ad un nodo *v′* adiacente ad esso, tale che *(v, v′) ∈* *E*. Per una sequenza di azioni *π = (a1, …, an)* e un agente *i* si denota con *πi[x]* la posizione in cui si troverà l’agente *i* dopo aver eseguito le prime *x* azioni in *π*, partendo dal nodo di *start* dell’agente, *s(i)*. Una sequenza di azioni *π* si definisce *single-agent plan* se e solo se, eseguendola a partire da *s(i)* l’agente arriva a trovarsi in *t(i)*, la sua posizione di *goal.* Una soluzione è costituita da un insieme di *k* piani, uno per ogni agente presente nell’istanza del problema.

Immagine che contiene schermata, testo, Carattere

Descrizione generata automaticamente

Figura 1: esempio di istanza di MAPF. Sono presenti sei agenti e ad ognuno è assegnato un nodo di *goal* (fonte: J. Li [[3](#riferimento_3)]).

Affinchè una soluzione sia valida, non devono verificarsi collisioni durante l’esecuzione dei *single-agent plan* di ogni agente; se ciò dovesse accadere, significherebbe che almeno in un *timestep* si verifica un *conflitto* tra due piani. Negli studi riguardanti MAPF svolti finora esistono diverse definizioni di conflitto; verranno qui di seguito elencate quelle più comunemente utilizzate, considerando due *single-agent plan πi* e *πj.*

* ***Vertex conflict.*** Un conflitto su un vertice tra *πi* e *πj* si verifica quando, secondo questi piani, gli agenti *i* e *j* dovrebbero occupare lo stesso vertice del grafo nel medesimo *timestep*. Formalmente, se e solo se esiste un *timestep x* per cui *πi[x]=* *πj[x]*.
* ***Edge conflict.*** Un conflitto su un arco tra *πi* e *πj* avviene quando, seguendo tali piani, gli agenti *i* e *j* dovrebbero attraversare lo stesso arco del grafo, nel medesimo *timestep*, percorrendolo lungo la stessa direzione. Formalmente, se e solo se esiste un *timestep x* tale che *πi[x]=* *πj[x]* e *πi[x + 1]=* *πj[x + 1]*.
* ***Following conflict.*** Questo tipo di conflitto si verifica tra *πi* e *πj* quando uno dei due agenti andrà ad occupare un vertice che era occupato dall’altro agente nel *timestep* precedente, ovvero se e solo se esiste un *timestep x* tale che *πi[x + 1]=* *πj[x]*.
* ***Cycle conflict.*** Un conflitto di questa tipologia si verifica tra un insieme di *single-agent plan πi,πi+1, … πj* se e solo se in uno stesso *timestep* ogni agente dell’insieme si sposta in un vertice precedentemente occupato da un altro agente, formando quindi un ciclo. Formalmente, esso avviene se e solo se esiste un *timestep x* tale che *πi[x + 1] = πi+1[x]*, *πi+1[x + 1] = πi+2[x]*, …, fino a *πj-1[x + 1] = πj[x] e πj[x + 1] = πi[x]*.
* ***Swapping conflict.*** Un conflitto di scambio si verifica tra due piani *πi* e *πj* quando, seguendoli, due agenti si dovrebbero scambiare di posizione. Formalmente, avviene se e solo se esiste un *timestep x* tale che *πi[x + 1] = πj[x]* e *πj[x + 1] = πi[x]*.

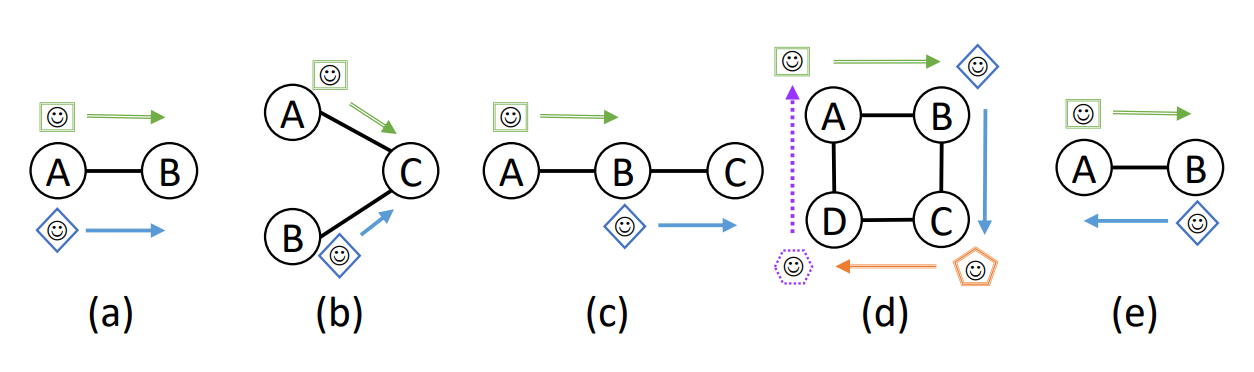


Figura 2: rappresentazioni grafiche delle più comuni tipologie di conflitto. Da sinistra a destra: *edge conflict*, *vertex conflict*, *following conflict*, *cycle conflict* e *swap conflict* (fonte: Stern et al. [[1](#riferimento_1)]).

In una soluzione per un’istanza di MAPF non è garantito che gli agenti giungano alle posizioni di *goal* nello stesso *timestep*, anzi è improbabile che ciò accada. Di conseguenza, è necessario stabilire il comportamento che dovranno seguire gli agenti una volta terminato il percorso assegnatogli e raggiunta la propria destinazione, mentre attendono che anche gli altri agenti terminino il loro movimento. Esistono principalmente due opzioni che possono essere scelte per il problema.

* ***Stay at target.*** Seguendo questo paradigma, una volta raggiunto il proprio nodo di *goal* un agente rimarrà fermo in quella posizione; un comportamento in pratica equivalente al compiere un’azione di *wait* ad ogni *timestep* successivo all’arrivo in quel vertice. Ciò significa che l’agente, pur avendo terminato il proprio percorso, potrà causare conflitti con altri agenti che dovessero transitare nel vertice da esso occupato. La soluzione dovrà tenere conto di questa possibilità ed essere costituita da piani nei quali tali conflitti non si verifichino.
* ***Disappear at target.*** In questo caso, quando un agente raggiunge il proprio nodo di *goal* è come se esso fosse rimosso dall’ambiente, in quanto esso non genererà più conflitti di alcun tipo a partire dal *timestep* successivo a quello di arrivo al *goal*.

Una volta definito chiaramente cosa costituisce un conflitto e quale comportamento devono avere gli agenti una volta arrivati al *goal*, resta da definire la funzione obiettivo. Per un’istanza di MAPF esistono nella maggior parte dei casi più soluzioni possibili, potenzialmente anche centinaia o migliaia, e alcune saranno ragionevolmente migliori di altre. Per poter confrontrare tra loro diverse soluzioni e valutarne correttamente la bontà, è necessario formalizzare una funzione obiettivo. Ne esistono diverse quelle illustrate di seguito sono tra le più frequentemente utilizzate.

* ***Sum of costs.*** Viene considerata come costo della soluzione la somma dei costi individuali di tutti i *single-agent plan* che la costituiscono. Il costo di un singolo piano equivale al numero totale di *timestep* richiesti all’agente per portarlo a termine. Formalmente, per una soluzione *π = {π1, …, πk}* il costo è definito come *∑1*≤i≤k *| πi|*.
* ***Makespan.*** Si considera come costo della soluzione il numero totale di *timestep* necessari affinchè tutti gli agenti arrivino alle rispettive destinazioni. Ciò equivale a considerare il massimo tra i costi dei singoli piani. Formalmente il costo per *π = {π1, …, πk}* è uguale a *max1*≤i≤k *| πi|*.
* ***Fuel.*** Funzione utlizzata meno frequentemente ma comunque presente nella letteratura (ad es. Felner et al. 2004 [[2](#riferimento_2)]), considera come costo della soluzione la somma dei costi dei *single-agent plan*, come per *sum-of-costs*, con però una differenza: non vengono considerate le azioni di *wait*. In questo modo il costo del piano per un agente aumenta solo quando esso effettivamente si muove, rappresentando il consumo di una risorsa legato al movimento, come ad esempio il carburante o la carica di una batteria.

Si può notare come, nel caso si scelga *disappear at target* come *agent-at-target behavior*, il costo della soluzione sia determinato chiaramente dalla funzione obiettivo scelta. Se invece si scegliesse *stay at target*, in associazione con la funzione *sum of costs*, si dovrà specificare in che modo varii il costo della soluzione in base al periodo di attesa degli agenti fermi nei loro nodi di *goal*. Un’assunzione comune è quella per cui la sosta di un agente conti come una serie di azioni di *wait* (*e quindi vada ad aumentare il costo totale*) a meno che esso non si muova più da quella posizione.

Quelle descritte sono le assunzioni comunemente utilizzate nella definizione dei problemi di MAPF, tuttavia esistono ulteriori possibilità che sono state proposte negli anni all’interno di diversi articoli scientifici, per adattare il problema alla rappresentazione di più contesti, a volte molto diversi tra loro. Queste varianti, che vanno oltre MAPF “classico”, meritano di essere presentate, in quanto il lavoro esposto nel qui presente elaborato potrebbe in futuro essere ampliato, incorporandone degli elementi.

2.2 Varianti di MAPF

Una prima modifica che è possibile apportare al modello “classico” di MAPF introdotto finora riguarda la struttura del grafo *G,* rappresentante l’ambiente di riferimento per gli agenti. Solitamente si assume che le azioni degli agenti richiedano sempre esattamente un *timestep* per essere completate, siano esse *move* o *wait*, ma ciò non è sempre il caso nella letteratura di MAPF, dove sono presenti modelli che si discostano da questo paradigma. Si possono definire azioni che richiedono tempi di esecuzione diversi, trasformando *G* in un grafo pesato, dove ad ogni arco viene assegnato un peso indicante il tempo necessario per il suo attraversamento. Un grafo di questo tipo esistente nelle ricerche svolte finora (ad es. Andreychuk et al. [[4](#riferimento_4)]) è una griglia *2k-neighbour*, dove, per ogni agente, le azioni *move* che può intraprendere sono limitate a i 2k vertici del grafo più vicini al nodo da esso occupato, dove *k* è un parametro. Impostando *k = 2* si ottiene il problema standard, dove l’agente può muoversi di un solo vertice in una delle quattro direzioni cardinali, mentre con valori di *k* maggiori di 2 si ottiene una maggiore libertà di movimento per gli agenti. In questo contesto, i costi delle azioni sono basati sulla distanza Euclidea tra il nodo occupato dall’agente e quello che andrà ad occupare una volta terminata l’azione scelta.

Immagine che contiene diagramma, linea, modello

Descrizione generata automaticamente

Figura 3: modelli di movimento per gli agenti, in griglie *2k-neighbour*. Il valore del parametro *k* è, da sinistra a destra, 2, 3, 4 e 5 (fonte: Andreychuk et al. [[4](#riferimento_4)]).

Un’altro aspetto del problema che può essere modificato riguarda le condizioni di ammissibilità di una soluzione. Viene comunemente richiesto solo che i *single-agent plan* che costituiscono la soluzione non producano dei conflitti, ma possono essere aggiunti ulteriori vincoli, per adattare il problema a situazioni dai requisiti più stringenti.

* **Regole di robustezza.** Requisiti di questo tipo impongono che la soluzione trovata per un’istanza del problema sia resistente a ritardi nell’esecuzione, che potrebbero verificarsi in seguito ad imprevisti. Un piano k-robusto è costruito in modo tale da poter sopportare ritardi di *k* *timestep* senza generare collisioni tra gli agenti. Come viene mostrato nella ricerca in [[5](#riferimento_5)], aggiungere questo requisito al problema ne aumenta la difficoltà, richiedendo tempi di risoluzione più lunghi, che vanno ad aumentare in relazione alla grandezza del parametro *k*. Tuttavia, utilizzare un piano robusto diminuisce la probabilità che, durante l’esecuzione dei piani, ritardi imprevisti generino delle collisioni e che si debbano quindi formulare nuovi percorsi per gli agenti. Di conseguenza, in contesti dove il movimento degli agenti può essere soggetto a ritardi, l’introduzione di regole di robustezza può portare a miglioramenti nelle prestazioni.
* **Regole di formazione.** Si possono imporre limitazioni al movimento degli agenti, richiedendo che la loro posizione sia in qualche misura collegata a quella degli altri automi presenti nel grafo. Per esempio, può essere richiesto che gli agenti si muovano rimanendo in una determinata formazione (ad es. [[6](#riferimento_6)]). La risoluzione del problema MAR si inserisce perfettamente in contesti di questo tipo, è infatti ragionevole pensare che, a causa di eventi imprevisti, agenti che debbano muoversi in formazione possano uscirne momentaneamente. In quel caso, il problema diventerebbe appunto operare una *reconnection*, per farli tornare alla formazione richiesta.

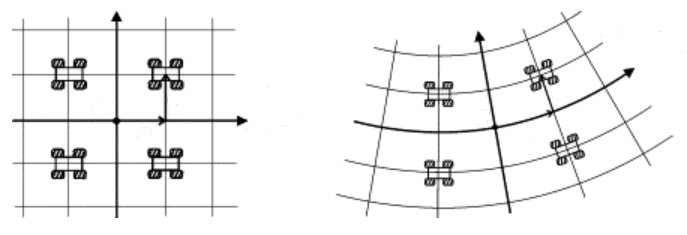


Figura 4: rappresentazione di quattro robot che si muovono mantenendo una formazione quadrata (fonte: Barfoot e Clark [[6](#riferimento_6)]).

Nella versione “classica” di MAPF ogni agente occupa solamente un vertice del grafo, questo perché un agente è un oggetto astratto. Nella realtà però, i robot che gli agenti rappresentano hanno una forma ed una dimensione e tali caratteristiche vanno considerate per poter garantire l’assenza di collisioni tra di essi. Esistono ricerche riguardanti MAPF che considerano agenti che variano tra loro per forma e dimensione (ad es. Li et al. [[7](#riferimento_7)]). Aggiungere questo livello di realismo al problema significa dover ripensare anche altri aspetti di esso, come ad esempio le definizioni di conflitto, infatti con agenti dotati di dimensioni possono verificarsi nuovi tipi di collisioni. Se un agente di grosse dimensioni occupa un vertice, esso potrebbe impedire ad altri agenti di attraversare nodi adiacenti ad esso. Similmente, un agente potrebbe collidere con un ostacolo, nonostante il percorso che sta seguendo comprenda solamente vertici liberi. Vengono quindi a crearsi nuovi tipi di conflitti: *vertex-to-vertex*, *edge-to-edge* e *edge-to-vertex* (Honig et al. [[8](#riferimento_8)]).

Altre ricerche su MAPF fanno avvicinare ulteriormente gli agenti del problema alla loro controparte reale, imponendo vincoli cinematici sul loro movimento (ad es. Hoenig et al. [[9](#riferimento_9)]). In queste varianti del problema le azioni disponibili per un agente non dipendono solamente dalla sua posizione (*ed eventualmente da quella degli altri agenti*) ma anche dalla sua velocità, dall’angolo di orientamento e/o altre variabili che possono essere considerate. Si può ad esempio imporre un limite alla velocità raggiungibile dall’agente, oppure all’angolo che può formare la loro traiettoria quando cambiano direzione, in questo modo si potranno rappresentare agenti che non sono in grado di effettuare virate brusche, per i quali bisogna dedicare più spazio di manovra in caso di una inversione di marcia.

Esistono altre varianti di MAPF che, per brevità, non verranno menzionate e, nel prossimo futuro, certamente ne verranno create di nuove, trattandosi di un problema la cui attenzione da parte del mondo accademico, specialmente negli ultimi anni, è notevole[[1]](#footnote-1). Avendo definito MAPF e alcune delle varianti attualmente esistenti, verranno ora presentati alcuni degli algoritmi che sono stati ideati per la sua risoluzione, in particolare CBS, vista la sua importanza all’interno di questa tesi.

2.3 Risoluzione di MAPF

La letteratura di MAPF dimostra come la risoluzione ottima del problema sia *NP-hard* [[10](#riferimento_10)]. Di conseguenza, non è facile sviluppare degli algoritmi che possano risolvere il problema in modo ottimo e allo stesso tempo efficiente. Negli anni sono state proposte diverse soluzioni, che si possono dividere principalmente in due categorie: algoritmi che risolvono il problema in modo ottimo, e algoritimi subottimali. Questi ultimi, sebbene possano essere incompleti e non trovare la soluzione migliore, spesso ne producono una in tempi migliori rispetto ad algoritmi ottimi e possono quindi essere validi strumenti, specialmente in contesti caratterizzati da una scarsa disponibilità di tempo per l’elaborazione dei piani per gli agenti.

Un esempio di algoritmo subottimale è *Prioritized Planning* [[11](#riferimento_11)]nel quale viene scelto un ordine di priorità tra gli agenti e, successivamente, viene calcolato un *single-agent plan* per ogni agente, seguendo l’ordine scelto. Si impone che il piano per un agente *ai* non possa avere conflitti con piani precedentemente generati per agenti con priorità maggiore. Se per un agente non è possibile trovare un percorso privo di conflitti, viene selezionato un nuovo ordine di priorità e la procedura viene ripetuta dall’inizio. L’algoritmo non è completo ed è inoltre subottimale, ma nella pratica trova soluzioni velocemente e con un elevato grado di ottimalità. A partire da questo algoritmo relativamente semplice, diversi ricercatori ne hanno sviluppati di più efficienti, andando a mitigarne gli aspetti negativi e ottenendo prestazioni migliori (ad es. Ma et al. [[12](#riferimento_12)]). Un’altra famiglia di algoritmi subottimali è detta *procedure-based* e comprende strategie che definiscono procedure, di cui si servono per risolvere i conflitti. Un esempio di algoritmo di questo tipo è *Push and Swap* [[13](#riferimento_13)] che introduce due primitive: *push* e *swap*. L’operazione *push* muove un agente verso il proprio *goal*, facendo spostare altri agenti che ne bloccano il percorso.Se esso arriva ad un punto in cui non può compieri ulteriori movimenti, per uscire dallo stallo viene utilizzata l’operazione *swap*, chepermette a due agenti di scambiarsi di posizione, senza alterare la configurazione degli altri agenti.

Tra gli algoritmi ottimali, si possono contare solutori che per risolvere MAPF lo riducono, trasformandolo in altri problemi, per i quali esistono già metodi conosciuti di risoluzione efficace, come *Integer Linear Programming* [[14](#riferimento_14)] o SAT [[15](#riferimento_15)]. Altri algoritmi sono invece *search-based* e puntano quindi a risolvere MAPF come un problema di *search*. La tecnica più semplice di questo tipo consiste nell’utilizzo di A\* (*algoritmo di search ottimo*) per effettuare una ricerca nello spazio degli stati del problema, definendo uno stato come un vettore di dimensione *k* (*con k uguale al numero di agenti*) delle posizioni di ogni agente all’interno del grafo. Lo stato iniziale del problema corrisponderà all’insieme delle posizioni di *start* degli agenti, mentre uno stato di *goal sarà* tale per cui ogni agente occupa il proprio nodo di *goal*. Questa ricerca è ottima ma, poiché la dimensione dello spazio degli stati cresce esponenzialmente all’aumentare degli agenti, non si tratta di un algoritmo praticabile, se non per istanze molto piccole del problema. Esistono comunque strategie volte a contenere il numero di stati, andando per esempio a ridurre il numero di agenti, dividendoli in gruppi indipendenti tra loro e risolvendo il problema separatamente per ogni gruppo [[16](#riferimento_16)].

Capitolo 3

Analisi del software sviluppato

Capitolo 4

Analisi degli esperimenti svolti

Bibliografia

[1] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, E. Boyarski, R. Bartak, “Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks” *Proceedings of the Symposium on Combinatorial Search (SoCS)*, pp.1-4,2019. Indirizzo: <http://mapf.info/index.php/Main.SKoen19k>

[2] A. Felner, R. Stern, S. Kraus, A. Ben-Yair, N. S. Netanyahu, “PHA\*: finding the shortest path with A\* in an unknown physical environment” Journal of *Artificial Intelligence Research Vol. 21*, pp. 631–670, 2004. DOI: <https://doi.org/10.1613/jair.1373>

[3] Jiaoyang Li, “Foundations of Multi-Agent Pathfinding”. Indirizzo: <https://jiaoyangli.me/research/mapf/>

[4] A. Andreychuk, K. Yakovlev, P. Surynek, D. Atzmon, R. Stern, “Multi-agent pathfinding with continuous time” *Artificial Intelligence Vol. 305*, 2022. DOI: <https://doi.org/10.1016/j.artint.2022.103662>

[5] D. Atzmon, R. Stern, A. Felner, G. Wagner “Robust Multi-Agent Pathfinding” *Symposium on Combinatorial Search (SoCS)*, 2018. Indirizzo: <https://www.researchgate.net/publication/327932903_Robust_Multi-Agent_Path_Finding>

[6] T. D. Barfoot, C. M. Clark, “Motion planning for formations of mobile robots”, *Robotics and Autonomous Systems Vol. 46*, pp. 65-78, 2004. DOI: <https://doi.org/10.1016/j.robot.2003.11.004>

[7] J. Li, P. Surynek, A. Felner, H. Ma, T. K. S. Kumar e S. Koenig, “Multi-Agent Path Finding for Large Agents” *Proceedings of the AAAI Conference on Artificial Intelligence* *33(01)*, pp. 7627-7634, 2019. DOI: <https://doi.org/10.1609/aaai.v33i01.33017627>

[8] W. Hoenig, J. Preiss, T. K. S. Kumar, G. S. Sukhatme, “Trajectory Planning for Quadrotor Swarms” *IEEE Transactions on Robotics 34(4)*, pp. 1-14, 2018. DOI: <http://dx.doi.org/10.1109/TRO.2018.2853613>

[9] W. Hoenig, T. K. S. Kumar, L. Cohen, H. Ma, H. Xu, N. Ayanian, S. Koenig, “Multi-Agent Pathfinding with Kinematic Constraints” *International Conference on Automated Planning and Scheduling*, 2016. Indirizzo: <https://www.researchgate.net/publication/308322653_Multi-Agent_Path_Finding_with_Kinematic_Constraints>

[10] J. Yu e S. LaValle, “Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs” *Proceedings of the AAAI Conference on Artificial Intelligence 27(1)*, pp. 1143-1149, 2013. DOI: <https://doi.org/10.1609/aaai.v27i1.8541>

[11] M. A. Erdmann e T. Lozano-Perez, “On multiple moving objects” *Algorithmica Vol. 2*, pp. 477–521, 1987. DOI: <https://doi.org/10.1007/BF01840371>

[12] H. Ma, D. Harabor, P. J. Stuckey, J. Li, S. Koenig, “Searching with Consistent Prioritization for Multi-Agent Pathfinding” *Proceedings of the AAAI Conference on Artificial Intelligence, 33(01)*, pp. 7643-7650, 2019. DOI: <https://doi.org/10.1609/aaai.v33i01.33017643>

[13] R. Luna, K. E. Bekris, “Push and Swap: Fast Cooperative Path-Finding with Completeness Guarantees” *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, 2011. DOI: <http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-059>

[14] J. Yu e S. LaValle, “Planning optimal paths for multiple robots on paths” *IEEE International Conference on Robotics and Automation*, pp. 3612-3617, 2013. DOI: <10.1109/ICRA.2013.6631084>

[15] P. Surynek, “Towards Optimal Cooperative Path Planning in Hard Setups through Satisfiability Solving” *Proceedings of the 12th Pacific Rim international conference on Trends in Artificial Intelligence*, 2012. DOI: <10.1007/978-3-642-32695-0_50>

[16] T. S. Standley, “Independence Detection for Multi-Agent Pathfinding Problems” *MAPF@AAAI*, 2012. Indirizzo: <https://trevorstandley.com/papers/ID_for_MAPP.pdf>

1. Come dimostrato dalla quantità di pubblicazioni riguardanti l’argomento, alcune delle quali si possono trovare raccolte presso il sito dedicato a MAPF: <http://mapf.info/index.php/Main/Publications> [↑](#footnote-ref-1)