Reconnection Multi-Agent Pathfinding (R-MAPF): generazione efficiente di percorsi verso punti di incontro per un gruppo di agenti

Tesi di:

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 *Reconnection Multi-Agent Pathfinding problem* (R-MAPF), 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 loro 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 gli algoritmi sviluppati 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 *Reconnection Multi-Agent Pathfinding problem*.

1.1 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 alcuni dei principali algoritmi che sono stati realizzati per risolverli, in particolare quello che è stato utilizzato anche in questo progetto. Il Capitolo 3 spiega il contributo teorico dell’elaborato ed illustra gli algoritmi sviluppati. Il Capitolo 4 raccoglie e analizza i dati ottenuti dagli esperimenti svolti. Nel Capitolo 5 vengono tratte le conclusioni finali sul lavoro effettuato e si discutono possibili sviluppi futuri.

Capitolo 2

Contesto e stato dell’arte

Come già accennato nell’introduzione, il problema R-MAPF 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 R-MAPF, 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 sette agenti e ad ognuno è assegnato un nodo di *goal[[1]](#footnote-1)*.

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]*.

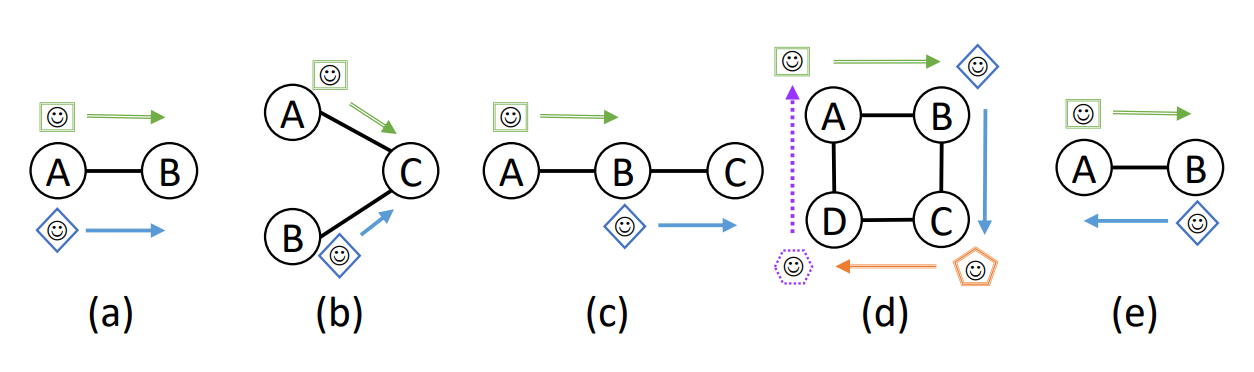


Figura 2: rappresentazione grafica delle più comuni tipologie di conflitto. Da sinistra a destra: *edge conflict*, *vertex conflict*, *following conflict*, *cycle conflict* e *swap conflict* (fonte: Stern et al. [[1](#riferimento_1)]).

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* superiori a 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 pari a, 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 R-MAPF 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 momentaneamente uscirne. In quel caso, il problema diventerebbe appunto operare una *reconnection*, per farli tornare alla formazione richiesta.

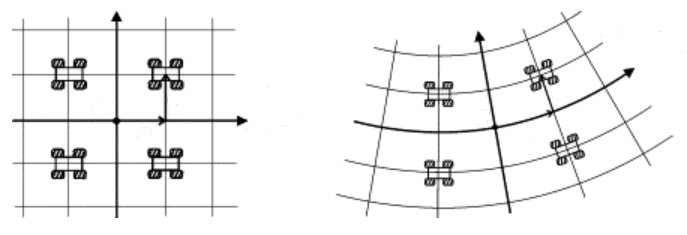


Figura 4: rappresentazione di quattro robot che si muovono mantenendo una formazione quadrata (fonte: Barfoot e Clark [[6](#riferimento_6)]).

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 loro. 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 ripensarne anche altri aspetti, 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 stia 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.

Due varianti degne di nota, che non modificano le assunzioni fatte sugli agenti o sui conflitti, quando piuttosto gli obiettivi del problema, sono *Permutation-Invariant MAPF* (PI-MAPF) [[21](#riferimento_21)] e *Shared-Goal MAPF* (SG-MAPF) [[22](#riferimento_22)]. In PI-MAPF le posizioni di *goal* fornite in input non sono preassegnate agli agenti; questo significa che i percorsi scelti devono portare ogni agente ad uno dei nodi di *goal*, ma non è importante, ai fini del problema, quale. SG-MAPF costituisce un caso speciale di PI-MAPF, in cui tutte le posizioni di *goal* sono identiche ed esiste quindi di fatto un unico nodo del grafo, fornito in input, a cui tutti gli agenti devono dirigersi.

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[[2]](#footnote-2). 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, in virtù della 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 ottimamente, 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 e da un elevato numero di 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 agentenon 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 *Boolean Satisfiability* (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\*, un algoritmo di *search* ottimo [[18](#riferimento_18)], 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 in un determinato *timestep*. Lo stato iniziale del problema corrisponderà all’insieme delle posizioni di *start* degli agenti, mentre uno stato di *goal* saràtale per cui ogni agente occuperà 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)].

Un altro algoritmo esistente, scelto in questa tesi per la sua ottimalità, unita all’efficienza che riesce a garantire, è *Conflict-Based Search* [[17](#riferimento_17)].

CBS produce una soluzione servendosi di due distinti livelli di ricerca. Nel livello alto, CBS opera una ricerca su un albero chiamato *constraint tree* (CT). Questo *albero dei vincoli* è un albero binario, in cui ogni nodo *N* è caratterizzato da:

1. Un’insieme di vincoli (*N.constraints*), ognuno dei quali relativo ad un singolo agente. Un vincolo è definito come una tupla *(ai, v, t)*, dove all’agente *ai* si vieta di occupare il vertice *v* durante il timestep *t*. La radice di CT contiene un insieme vuoto di vincoli, mentre il figlio di un nodo eredita i vincoli del nodo padre, ai quali ne viene aggiunto uno nuovo.
2. Una soluzione (*N.solution*), ovvero un insieme di *k single-agent plan*, uno per ogni agente. Tali piani verranno trovati dalla ricerca di basso livello. Il piano di ogni agente non deve violare i vincoli imposti su quell’agente.
3. Il costo totale della soluzione corrente (*N.cost*).

Un nodo *N* in CT viene considerato un nodo di *goal* se i percorsi degli agenti non presentano conflitti tra loro. La ricerca di alto livello opera una ricerca *best-first* nel CT, dove i nodi sono ordinati in base ai loro costi, per scegliere il prossimo nodo da espandere. A partire da un nodo *N* di CT, viene invocata la ricerca di basso livello. Essa genera il percorso di lunghezza minima per ogni agente, facendo in modo che esso non violi i vincoli imposti su quell’agente. Successivamente, i percorsi trovati vengono validati. Se non è presente nemmeno un conflitto tra i piani per gli agenti, il nodo *N* viene dichiarato nodo di *goal* e la soluzione corrente (*N.solution*) viene restituita. Se viene invece individuato un conflitto tra due o più agenti la validazione viene interrotta e *N* viene marchiato come nodo *non-goal*. Un conflitto si può rappresentare come una tupla *(ai, aj, v, t)* dove *ai* e *aj* sono due agenti, che occupano il vertice *v* nello stesso *timestep t*. Per risolvere il conflitto, vengono generati in CT due nuovi nodi figli a partire da *N*: il primo conterrà un nuovo vincolo *(ai, v, t)* e il secondo un nuovo vincolo *(aj, v, t)*. Imponendo che solo uno dei due agenti coinvolti possa occupare quel vertice nel *timestep* in questione, si risolve il conflitto e considerando entrambe le possibilità si garantisce l’ottimalità, in quanto tutte le possibili alternative vengono esplorate.

Immagine che contiene schizzo, cerchio, bianco, diagramma

Descrizione generata automaticamenteImmagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamente

Figura 5: (*a sinistra*) istanza di MAPF con due agenti e (*a destra*) *constraint tree* da essa derivato [[17](#riferimento_17)].

Come stabilito precedentemente, la ricerca di basso livello, dato un agente *ai* e un insieme di vincoli ad esso associati, si occupa di trovare un percorso di costo minimo per quell’agente che rispetti i vincoli, ignorando gli altri agenti. Per questo scopo può essere utilizzato qualsiasi algoritmo di *pathfinding* per singolo agente; spesso viene impiegato A\*.

Verrà ora introdotto formalmente il problema di *Multi-Agent Meeting* [[19](#riferimento_19)], [[20](#riferimento_20)], la cui generalizzazione, nella forma di *Reconnection Multi-Agent Pathfinding*, è oggetto di questa tesi.

2.4 *Multi-Agent Meeting* (MAM)

Il problema condivide alcuni aspetti di MAPF; in entrambi, infatti, sono presenti *k* agenti disposti su un grafo ed è necessario calcolare dei percorsi per guidarne il movimento. In MAPF per ogni agente è indicata nell’input, oltre ad una posizione di partenza, anche una posizione di arrivo, mentre in MAM tutti gli agenti devono convergere verso un’unica posizione di *meeting*, un *goal* condiviso tra gli agenti. Si può notare come MAM sia in particolare simile alla variante SG-MAPF. La differenza principale consiste nel fatto che tale posizione in MAM deve essere individuata e non fa parte dell’input. Questo rende i due problemi, sebbene simili, fondamentalmente distinti.

Un’istanza di MAM [[20](#riferimento_20)] prevede un input costituito da una tupla *(G, s)*, 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*. Una soluzione è costituita da un nodo *m ∈ V*, che corrisponde alla posizione di incontro scelta, e da un insieme di percorsi *π*, che conterrà un percorso per ogni agente, dalla posizione di partenza *si* alla posizione di *meeting m*. La versione base di MAM è detta *conflict-tolerant* e permette la presenza di agenti che occupino lo stesso vertice o che attraversino lo stesso arco nel medesimo momento. Di conseguenza, le azioni di *wait* non hanno ragione d’essere, in quanto per ogni agente si può calcolare il percorso più breve per arrivare alla posizione di incontro, ignorando completamente gli altri agenti. Nella versione *conflict-free* (CF-MAM) invece, come in MAPF, le collisioni tra agenti sono vietate e i percorsi scelti devono essere privi di conflitti. In questo scenario, gli agenti hanno a disposizione azioni di *move* e di *wait*. La versione del problema tollerante verso i conflitti è utile per rappresentare situazioni nelle quali più agenti possono coesistere nella medesima posizione, ad esempio quando i nodi del grafo rappresentano edifici mentre gli agenti sono persone che possono stare al loro interno. Quando invece la dimensione degli agenti è tale per cui due o più di loro non possono occupare lo stesso vertice, è opportuno formulare il problema servendosi di CF-MAM. Per calcolare il costo di una soluzione si possono utilizzare le funzioni obiettivo già introdotte per MAPF; solitamente viene utilizzata *sum of costs* oppure *makespan*.

Immagine che contiene testo, schermata, modello

Descrizione generata automaticamente

Tabella 1: panoramica dei problemi menzionati, messi a confronto per evidenziarne similitudini e differenze (fonte: Atzmon et al. [[20](#riferimento_20)]).

La risoluzione di CF-MAM è strettamente legata a quella di MAPF. In [[20](#riferimento_20)] sono stati proposti a tale scopo due algoritmi, entrambi ottimi e completi. Il primo, chiamato CF-MAM CBS opera una ricerca su due livelli, servendosi del framework di CBS. Il secondo, chiamato *Iterative Meeting Search*, risolve il problema come un’istanza di SG-MAPF, considerando iterativamente diverse posizioni di incontro, fino ad individuare quella ottima.

In conclusione, sono stati presentati alcuni dei più importanti ed interessanti problemi di *pathfinding* attualmente esistenti in letteratura: MAPF, con alcune delle sue varianti, e MAM. Il presente elaborato si inserisce in un contesto accademico, quello dei problemi di *pathfinding* per molteplici agenti, per il quale nel corso degli anni è stata svolta una grande quantità di ricerca da parte di numerosi studiosi. Grazie a questa ricerca preesistente che ha definito, formalizzato e risolto gli aspetti principali dei problemi di questa natura, è possibile la realizzazione di questa tesi, che va ad esplorare una possibile estensione di MAM, servendosi di un algoritmo sviluppato per MAPF, ovvero CBS.

Capitolo 3

Analisi del lavoro svolto

In questo Capitolo verrà illustrato dettagliatamente il lavoro effettuato in questa tesi, comprensivo dei risultati raggiunti e degli algoritmi sviluppati. È innanzitutto necessario introdurre formalmente il problema affrontato.

3.1 *Reconnection Multi-Agent Pathfinding* (R-MAPF)

Sebbene MAM possa essere un valido strumento, utile al fine di rappresentare e risolvere diverse situazioni, il fatto che gli agenti debbano incontrarsi in un unico vertice del grafo può essere una limitazione. Esistono contesti nei quali più agenti non possono occupare lo stesso nodo del grafo ed in questi casi il problema risulta difficile da rappresentare. R-MAPF viene introdotto in questo elaborato per sopperire a questa mancanza.

Un’istanza di R-MAPF con *k* agenti prevede un input costituito da una tupla *(G, s)*, 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*. Oltre al grafo *G*, che può descritto come il grafo primario del problema, deve essere presente anche un secondo grafo *C*, il grafo di connettività. *C = (V, E′)* è un grafo non orientato, nel quale *V* è lo stesso insieme di nodi presenti in *G*, mentre *E′* ⊆ *E* è l’insieme degli archi, in cui un arco tra due vertici esiste se e solo se i due vertici sono connessi. La connettività è una qualità che può avere diversi significati: può ad esempio rappresentare il fatto che due agenti, mentre occupano due nodi connessi, possano comunicare tra di loro, oppure che essi possano vedersi chiaramente. Una soluzione ad un’istanza di R-MAPF è costituita da un sottoinsieme di *V*, *V′*,contenente esattamente *k* nodi di *goal* e da un insieme di percorsi *π*, che contiene per ogni agente *i* un percorso che, se seguito, lo condurrà dal suo nodo di partenza ad una delle posizioni facenti parte di *V′*. I nodi contenuti in *V′* dovranno essere una cricca,ossia unsottoinsieme fortemente connesso, all’interno di *C*. Ciò significa che, presi due nodi qualsiasi *vi* e *vj* in *V′*, dovrà esistere in *E′* un arco che li colleghi. I percorsi scelti non dovranno generare conflitti tra loro, di conseguenza seguendoli gli agenti non dovranno subire collisioni tra di loro o con ostacoli presenti nell’ambiente.

Immagine che contiene diagramma, linea, cerchio, modello

Descrizione generata automaticamente

Figura 6: (*a sinistra*) esempio di grafo primario *G* di un’istanza di R-MAPF; (*a destra*) grafo di connettività *C* che potrebbe essere associato a *G*. Si può notare come siano presenti in *C* cricche di due, tre e quattro nodi. Se l’istanza del problema avesse cinque agenti, non esisterebbe una soluzione.

In questo elaborato si utilizzano per R-MAPF le stesse assunzioni di MAPF “classico”. Il tempo è discreto e in ogni *timestep* ogni agente occupa esattamente un vertice del grafo. Gli agenti hanno tutti la stessa forma e dimensione e ad ogni *timestep* hanno a disposizione le due azioni convenzionali: *move*, con la quale un agente può muoversi di esattamente una posizione in una delle quattro direzioni cardinali,e *wait*. I conflitti che si richiede siano assenti in una soluzione valida sono i *vertex*, *edge* e *swap conflicts*. In questo modo si impedisce che più agenti possano occupare lo stesso vertice o attraversare lo stesso arco nello stesso momento. Per il comportamento degli agenti al raggiungimento del *goal* viene utilizzata l’assunzione *stay at target*; questa scelta è dettata dalla natura del problema, in quanto, poiché gli agenti devono incontrarsi in posizioni scelte affinchè possano comunicare tra loro, è logico che essi debbano attendere nei rispettivi nodi di *goal* l’arrivo degli altri agenti. Fino a che tutti gli agenti non hanno raggiunto le proprie destinazioni, il problema non è risolto ed è quindi corretto che, calcolando i percorsi, si considerino come potenziali fonti di conflitti anche gli agenti che hanno già completato il proprio piano e stanno attendendo al *goal*.

Immagine che contiene schermata, diagramma, Policromia, cerchio

Descrizione generata automaticamente

Figura 7: esempio di istanza di R-MAPF con otto agenti. Sono già stati trovati i nodi di *goal* per gli agenti, per i quali sono evidenziati gli archi che li collegano nel grafo *C* di connettività.

La funzione obiettivo utilizzata per valutare la qualità di una soluzione è *sum of costs*. È stata scelta in quanto risulta perfettamente adatta al problema affrontato ed anche in virtù del fatto che è già stata ampliamente utilizzata in lavori precedenti riguardanti MAPF e MAM. In questo modo il lavoro svolto si allinea con la letteratura già esistente riguardante gli algoritmi di *pathfinding*.

In questo elaborato si vuole anche, tuttavia, introdurre una variante di *sum of costs* che, a conoscenza dell’autore, non è stata utilizzata precedentemente. Come nella versione classica della funzione, il costo di una soluzione viene calcolato sommando i costi di tutti i *single-agent plan* in essa contenuti. La novità introdotta consiste nel fatto che, sebbene le azioni di *wait* vengano conteggiate tanto quanto quelle di *move*, le azioni di *wait* intraprese da un agente che sta occupando il suo nodo di *goal* non vengono considerate. L’assunzione comune, quando viene utilizzata *sum of costs* in congiunzione con *stay at target*, è che le azioni di *wait* di un agente continuino ad avere costo unitario anche quando esso si trova già a destinazione, a meno che sia previsto dal suo piano che esso non si muova più da quella posizione. Introducendo questa modifica, alcune soluzioni che prima sarebbero risultate ugualmente buone avranno valori diversi. Per mostrare ciò, verrà utilizzata l’istanza rappresentata nella Figura 8 come riferimento, nella quale sono già stati individuate le posizioni di *goal* a cui devono dirigersi gli agenti. Un esempio di insieme di percorsi validi comprende (S1, G1), (S2, S2, S2, S2, G2), (S3, c, d, G2, G3); la soluzione caratterizzata dai *goal* scelti e da questi percorsi verrà identificata come *soluzione A*. Un altro insieme di percorsi validi comprende (S1, G1), (S2, G2, G2, S2, G2), (S3, c, d, G2, G3); la soluzione caratterizzata dai *goal* scelti e da questi percorsi verrà identificata come *soluzione B*. Si può notare come, utilizzando la funzione obiettivo *sum of costs*, i costi delle due soluzione siano identici, 9 per entrambe. Utilizzando invece la variante della funzione discussa precedentemente, la *soluzione A* risulta avere costo pari a 9, mentre la *soluzione B* ha costo pari a 8 e verrebbe quindi scelta tra le due, avendo un costo strettamente minore rispetto ad *A*. Grazie a questa nuova variante di *sum of costs*, si prediligono quindi soluzioni che porterebbero un agente ad arrivare prima al suo *goal*, anche se poi dovesse momentaneamente spostarsi per far passare un altro agente. In questo modo, in virtù dell’obiettivo del problema, ossia riunire un gruppo di agenti, si fa in modo che almeno un sottoinsieme degli agenti possa riconnetersi il prima possibile. Nell’esempio esposto, seguendo la *soluzione* *B* gli agenti 1 e 2 sarebbero in condizione di poter comunicare già dai primi *timestep*, mentre nella *soluzione A* ciò non accade. L’esempio utilizzato, per semplicità, contiene un grafo di piccole dimensioni, ma su mappe più grandi l’effetto della variazione alla funzione obiettivo diviene ancora maggiore. Sebbene questa variante di *sum of costs* non sia stata utilizzata per gli esperimenti svolti, avendo voluto preferire la versione standard, sulla base del suo attestato largo utilizzo in letteratura, essa potrebbe risultare utile in determinati contesti.

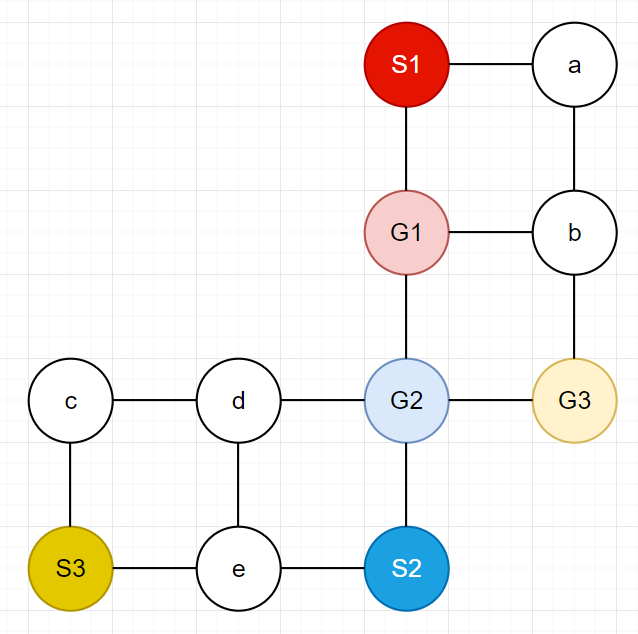


Figura 8: esempio di istanza di R-MAPF con tre agenti. I nodi S1, S2 e S3 corrispondono ai nodi di partenza degli agenti, mentre G1, G2 e G3 sono i nodi di *goal*. Si assuma che i nodi di *goal* formino una cricca all’interno del grafo *C* di connettività.

Avendo definito formalmente il problema e avendo dichiarato le assunzioni utilizzate, è ora possibile discuterne la risoluzione.

3.2 Risoluzione di R-MAPF

Trovare una soluzione per un’istanza di R-MAPF con *k* agenti richiede fondamentalmente due passaggi: innanzitutto è necessario individuare un insieme di *k* vertici che costituisca una cricca all’interno del grafo di connettività *C* legato all’istanza, essi saranno i nodi di *goal* per gli agenti; successivamente dovranno essere calcolati dei percorsi privi di conflitti che possano condurre gli agenti ai *goal* scelti. Ai fini del problema, non è importante in quale nodo di *goal* si diriga ogni agente, in quanto l’obiettivo consiste nel riunire gli agenti nell’insieme di vertici scelti, per poterli riconnettere, e a tale scopo è sufficiente che ogni posizione di *goal* designata sia occupata esattamente da uno degli agenti. Di conseguenza, una volta determinato l’insieme di nodi di *goal*, è possibile decidere un assegnamento, anche arbitrario, che faccia corrispondere ad ogni agente uno specifico vertice di *goal*. Qualunque assegnamento agente-*goal* sarà valido proprio perché il problema richiede solamente che gli agenti, una volta terminati tutti i loro percorsi, si trovino ad occupare la cricca di nodi scelta. A questo punto, è possibile risolvere l’istanza come se si trattasse di un’istanza di MAPF; sono infatti presenti *k* agenti, ognuno con una posizione di partenza ed una di arrivo, ed è necessario calcolare per loro percorsi privi di conflitti che li conducano a tali posizioni del grafo. Dunque, per calcolare i percorsi degli agenti è possibile utilizzare CBS. L’algoritmo è completo, ottimo e richiede molto spesso poco tempo per calcolare i percorsi [[17](#riferimento_17)]; tali motivi lo rendono una scelta ragionevole per risolvere le istanze di R-MAPF.

Immagine che contiene diagramma, linea, schermata, cerchio

Descrizione generata automaticamente

Figura 9: rappresentazione delle diverse fasi di risoluzione di un’istanza di R-MAPF. (*a sinistra*) grafo in cui sono evidenziati i nodi di partenza dei due agenti, S1 e S2; (*al centro*) sono stati determinati i nodi di *goal*; (*a destra*) è stato effettuato l’assegnamento di ogni *goal* ad un determinato agente, l’istanza del problema è a questo punto risolvibile con CBS.

Nel paragrafo precedente si è descritto come è possibile trovare una soluzione ad un’istanza di R-MAPF. L’obiettivo preposto, tuttavia, non è solamente calcolare una soluzione valida, ma fare in modo che essa sia anche ottima. A questo scopo, al posto di utilizzare un assegnamento arbitrario per gli agenti, è opportuno indirizzare ogni agente al nodo di *goal* che gli è più semplice da raggiungere, in modo da poter ridurre il più possibile il costo complessivo dei percorsi degli agenti. Il fattore che, tuttavia, spesso determina maggiormente il costo di una soluzione è la scelta dell’insieme di nodi di *goal*.

Per determinare i vertici del grafo in cui dovranno dirigersi gli agenti per potersi riconnettere sono disponibili diverse strategie. Il metodo ottimo più elementare consiste nell’individuare tutte le cricche di dimensione *k* presenti nel grado di connettività del problema, con *k* uguale al numero degli agenti presenti, per poi scegliere quella migliore, ossia la cricca di nodi che richiede il minor costo possibile per essere raggiunta ed occupata dagli agenti. Per operare questa scelta occorre determinare per ogni cricca il costo che avrebbe la soluzione se gli agenti dovessero riunirsi in tale insieme di nodi. Ognuno di questi costi può essere calcolato scegliendo l’assegnamento agente-*goal* ottimo per la cricca in questione (*successivamente verrà illustrato come*) e in seguito calcolando i percorsi ottimi con CBS; il costo cumulativo dei percorsi trovati corrisponderà al costo associato alla cricca. L’ottimalità di questo algoritmo è facilmente dimostrabile, in quanto vengono analizzate in modo esaustivo tutte le possibilità presenti nell’istanza del problema, tuttavia è altrettanto semplice comprendere come esso sia totalmente impraticabile a causa dei tempi richiesti per la sua esecuzione. Scovare tutte le cricche di nodi di dimensione *k* presenti nel grafo di connettività è un’operazione già di per sé dispendiosa, ma dover eseguire CBS per un numero di istanze potenzialmente nell’ordine delle centinaia o delle migliaia può richiedere tempi enormi.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Algoritmo 1: ricerca di tutte le cricche di dimensione *k* presenti all’interno di un grafo di connettività *C*.

Un metodo più interessante per trovare una cricca di nodi nel grafo di connettività consiste nel generarla. È infatti possibile *costruire* l’insieme dei *goal* un nodo alla volta, partendo da una cricca di dimensione inferiore a *k* ed espandendola aggiungendo iterativamente un nodo che sia connesso, nel grafo di connettività, a tutti gli altri nodi già presenti nella cricca. In questo modo si ottiene infine una cricca della dimensione necessaria per risolvere l’istanza del problema. L’algoritmo sviluppato in questo elaborato per risolvere il problema di R-MAPF opera esattamente in questo modo. La ricerca che esso effettua si può raffigurare con una struttura ad albero, in cui ogni nodo contiene una cricca. I nodi dell’albero che si trovano alla stessa profondità conterranno cricche di dimensioni uguali e la profondità massima dell’albero sarà uguale a *k* in quanto, anche se esistessero cricche di dimensioni maggiori nel grafo di connettività dell’istanza considerata, non servirebbero e non verrebbero quindi generate. La radice conterrà una *cricca vuota* ed avrà una serie di nodi figli al livello 1 dell’albero; ognuno di loro conterrà una *cricca banale* formata da un solo vertice e sarà presente uno di questi nodi per ogni vertice esistente nel grafo di connettività. L’algoritmo, a partire dal primo dei nodi del primo livello, effettua una ricerca in profondità. Ad ogni iterazione, esso espande un nodo dell’albero in questo modo: vengono considerati i vertici che sono connessi a tutti i vertici facenti parte della cricca, ma che non ne fanno ancora parte; per ognuno di loro viene generato un nodo figlio, contenente la cricca formata dalla cricca del nodo padre, con in aggiunta il nuovo vertice. Trattandosi appunto di una ricerca in profondità, all’iterazione successiva l’algoritmo andrà ad espandere il primo dei nuovi nodi generati. Può succedere che non sia possibile espandere un nodo, poiché non esistono vertici che si possano aggiungere alla cricca ad esso associata. In questo caso, affinchè la ricerca non si blocchi, è necessaria un’azione di *backtracking*: la ricerca dovrà ripartire dal nodo successivo del livello corrente; se non ci fossero altri nodi, si dovrà risalire nell’albero di un livello. Eseguendo l’algoritmo descritto sono possibili due scenari:

* espandendo un nodo di livello *k-1* si genera un nodo al livello *k* che contiene una cricca di *k* vertici. Essa viene restituita e l’algoritmo termina.
* vengono espansi tutti i nodi dell’albero ma non viene mai generato un nodo al livello *k*. L’istanza del problema non ha soluzione.

Nella Figura 10 è raffigurato un esempio di esecuzione dell’algoritmo. Il grafo di connettività utilizzato è quello della Figura 6. Si assuma che l’istanza del problema in esame abbia quattro agenti. Come si può vedere, al primo livello dell’albero sono presenti nodi contenenti i singoli vertici del grafo, che nella figura non sono rappresentati nella loro totalità per mancanza di spazio. La ricerca parte dal primo nodo dell’albero, contenente (a). Espandendolo viene generato il nodo contenente la cricca (a, b). Quest’ultimo nodo non può essere espanso e al momento non sono presenti altri nodi a quel livello; la ricerca riparte dal livello precedente e viene espanso il nodo contenente (b). L’algoritmo continua ad espandere nodi ed effettuare *backtracking* quando raggiunge punti di stallo, fino ad espandere il nodo contenente (e). A partire dall’espansione di quel nodo, la ricerca porta infine alla generazione del nodo contenente la cricca (e, f, h, i); essa è una cricca della dimensione cercata e di conseguenza viene restituita, terminando la ricerca.

Immagine che contiene diagramma, disegno, schizzo, linea

Descrizione generata automaticamente

Figura 10: rappresentazione di una sezione dell’albero generato dall’algoritmo.

L’algoritmo descritto è sempre in grado di trovare un cricca di dimensione *k* in un grafo di connettività, se in esso ne esiste almeno una. Non è tuttavia garantito che la cricca restituita sia quella ottima, poiché appena ne viene generata una della dimensione cercata l’algoritmo termina e non vengono quindi considerate altre cricche potenzialmente migliori. Se però si dovessero generare tutte le cricche di dimensione *k* presenti nel grafo e si dovessero confrontare tra di loro, l’algoritmo diventerebbe una ricerca esaustiva, non dissimile da quella descritta precedentemente, e sarebbe del tutto inefficiente. Sebbene l’algoritmo sia dunque subottimale, è possibile applicare delle migliorie per aumentare la qualità dei risultati ottenuti. Per come è stata descritta finora, la ricerca svolta dall’algoritmo è non informata: esso inizia ogni iterazione espandendo il primo nodo del livello a cui si trova. Il modo in cui si sviluppa la ricerca dipende quindi largamente dall’ordine in cui sono disposti i nodi contenenti le cricche, che al momento è del tutto arbitrario. Sarebbe invece opportuno riordinare i nodi, facendo in modo che ad ogni iterazione l’algoritmo espanda quello più *promettente*, sfruttando la conoscenza del problema e rendendo quindi la ricerca informata. Un’euristica che è possibile utilizzare, scelta in questo elaborato, consiste nel valutare le cricche contenute nei nodi dell’albero, in base alla distanza euclidea che i vertici che ne fanno parte hanno rispetto ai vertici di partenza degli agenti. L’euristica si basa sull’intuizione logica per cui, qualora gli agenti si incontrassero in un insieme di vertici vicini ai loro nodi di partenza, essi dovrebbero percorrere una distanza minore, rispetto ad un insieme di *goal* posizionati lontano da tutti gli agenti. Diminuendo la distanza media che dovrà percorrerre un singolo agente, il costo totale della soluzione sarà minore. Il valore *h* associato ad una cricca *c* dall’euristica descritta può essere definito formalmente con la seguente formula:

Dove *n* è un nodo facente parte di *c*, mentre *S* è l’insieme dei nodi di *start* degli agenti. Scelta un’euristica, è necessario utilizzarla per riordinare i nodi di un livello dell’albero ogni volta che ne vengono generati di nuovi a quel livello, in questo modo l’algoritmo selezionerà sempre per primo il nodo contenente la cricca considerata migliore secondo l’euristica. Come verrà mostrato nel Capitolo successivo tramite l’evidenza empirica raccolta, questo algoritmo è molto efficiente, richiedendo infatti tempi di esecuzione molto bassi. Inoltre, pur essendo subottimale, utilizzando l’euristica suggerita esso restituisce in media cricche che portano a soluzioni decisamente buone, non lontane dall’ottimo.

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

Algoritmo 2: ricerca informata di una cricca di dimensione *k* in un grafo di connettività *C*.

Una volta determinato l’insieme di nodi che dovranno essere occupati dagli agenti, è necessario scegliere per essi un assegnamento agente-*goal*. Volendo fare la scelta migliore, si tratta di trovare l’assegnamento che porti ad avere il minor costo complessivo possibile per i percorsi degli agenti. La situazione descritta non è altro che un’istanza del problema dell’assegnamento: esso richiede di associare coppie di elementi presi da due insiemi di uguale dimensione, considerando un costo associato ad ogni coppia e volendo minimizzare il costo totale. In questo caso i due insiemi sono l’insieme di nodi di partenza degli agenti e l’insieme dei nodi di *goal*, mentre il costo associato ad ogni coppia corrisponde al costo necessario perché un agente attraversi il percorso che li collega. Il problema è risolvibile in modo ottimo tramite un metodo noto come algoritmo ungherese [[23](#riferimento_23)]. L’unica criticità risiede nel fatto che in questo caso non è noto il costo associato ad ogni coppia; per conoscere con esattezza il costo del percorso tra la posizione di partenza di un agente ed uno dei nodi di *goal*, si dovrebbe infatti prima risolvere il problema, generando i percorsi privi di conflitti con CBS. Solo allora si potrebbe determinare il costo reale di un percorso, avendo tenuto conto degli altri agenti coinvolti nel problema. Quello che invece è possibile fare è assegnare ad ogni coppia un costo calcolato tramite un’euristica. A tale scopo, si può utilizzare l’algoritmo A\* per conoscere il costo del percorso più breve tra le due posizioni. Tale valore costituirà un *lower bound* del costo reale, in quanto un percorso tra le due posizioni trovato da CBS potrà avere solo costo uguale o maggiore a quello trovato da A\*. Ciò è corretto poiché il percorso trovato da A\* è quello ottimo, ma non tiene conto degli altri agenti, mentre quello trovato da CBS potrebbe essere più lungo, se dovessero essere inserite azioni di *wait* o deviazioni dal tragitto ottimo per scongiurare conflitti. Utilizzando l’algoritmo ungherese e servendosi di A\* come euristica, è dunque possibile determinare un assegnamento per gli agenti e di conseguenza risolvere in seguito l’istanza del problema con CBS.

In conclusione, in questo Capitolo è stato definito il problema di *Reconnection Multi-Agent Pathfinding* ed è stato proposto un metodo per risolverlo. Sono inoltre stati presentati algoritmi volti ad implementare tale metodo, che permettono di risolvere il problema in modo subottimale. Nel Capitolo seguente verrà mostrato, tramite i dati raccolti dagli esperimenti, come tali algoritmi siano efficienti e come le soluzioni trovate si avvicinino mediamente all’ottimo.

Capitolo 4

Analisi degli esperimenti svolti

Avendo formulato degli algoritmi volti a risolvere il problema introdotto, è necessario testarli per verificarne la validità. Tramite gli esperimenti svolti, esplicati in questo Capitolo, sono stati raccolti dati che permettono di avere una misura tangibile dell’efficienza degli algoritmi e dell’ottimalità delle soluzioni trovate. Verranno inoltre dichiarati gli strumenti e le tecnologie impiegate per lo sviluppo del software utilizzato per la realizzazione degli esperimenti.

4.1 Ambiente di sviluppoe risorse utilizzate

Il software sviluppato è stato interamente realizzato nel linguaggio di programmazione Python. Oltre ad essere un linguaggio di facile comprensione e con una grande quantità di supporto e di librerie che ne facilitano l’uso, questa scelta è stata presa anche per poter utilizzare una libreria già esistente, la quale implementa l’algoritmo CBS[[3]](#footnote-3). In questo modo non è stato necessario scrivere una parte di codice, fondamentale per il progetto ma allo stesso tempo caratterizzata da una discreta mole e complessità. Questa libreria è stata utilizzata senza apportare modifiche di alcun tipo, eccezione fatta per la sezione di codice riguardante la funzione obiettivo, nella quale è stata implementata la funzione *sum of costs* ed è stata aggiunta anche un’opzione che permette di utilizzare al posto di essala variante descritta precedentemente. L’IDE (*Integrated Development Environment*) utilizzato è Visual Studio Code[[4]](#footnote-4).

4.2 Definizione e preparazione degli esperimenti

Per poter realizzare gli esperimenti svolti, è stato innanzitutto necessario creare un programma che implementasse gli algoritmi sviluppati. È stato a questo scopo realizzato un solutore; esso è in grado di prendere in input un’istanza di R-MAPF, espressa sotto forma testuale e contenuta in un file, risolverla e restituire la soluzione, su *standard output* oppure salvandola in un file di testo. Per permettere il funzionamento del solutore, sono state create delle librerie che si occupano di implementare gli algoritmi necessari alla risoluzione del problema. Oltre all’algoritmo di generazione di una cricca tramite ricerca su albero, è stato necessario implementare, al fine degli esperimenti, anche la ricerca esaustiva di tutte le cricche di dimensione *k* presenti in un grafo di connettività. Per l’algoritmo ungherese, necessario per individuare l’assegnamento agente-*goal* ottimale, è stata invece utilizzata una libreria preesistente[[5]](#footnote-5). Oltre al solutore sono stati realizzati alcuni piccoli programmi, necessari per funzioni secondarie quali la generazione automatizzata delle istanze del problema e l’elaborazione dei dati raccolti.

Gli esperimenti svolti consistono fondamentalmente nella risoluzione di numerose istanze del problema tramite il solutore descritto. Sono state generate centinaia di istanze diverse, che variano tra loro per dimensione del grafo, densità e posizionamento degli ostacoli, per poi risolverle, in modo da raccogliere dati significativi sull’efficienza e l’efficacia degli algoritmi proposti.

Il primo esperimento svolto consiste nella verifica dell’utilità dell’euristica proposta per la generazione informata di una cricca, basata sulla distanza euclidea presente tra i vertici facenti parti della cricca e i nodi di *start* degli agenti. Si è voluto appurare se l’euristica scelta portasse effettivamente ad un miglioramento della qualità delle soluzioni trovate e se ciò avesse o meno un impatto sui tempi di esecuzione richiesti. L’esperimento ha inoltre l’obiettivo di determinare di quanto le soluzioni trovate dall’algoritmo, che è subottimale, si avvicinino in media all’ottimo. Per ottenere questo, si è fatto in modo che, per ogni istanza del problema affrontata dal solutore, venissero generate tutte le cricche, di dimensione pari al numero di agenti, esistenti nel grafo di connettività ad essa associato. Successivamente, per ogni insieme di nodi di *goal* trovato è stato determinato l’assegnamento migliore tramite l’algoritmo ungherese e l’istanza è stata risolta utilizzando CBS. In questo modo è stato possibile valutare tutti i possibili insiemi di nodi di *goal* che si potessero scegliere in un’istanza, in particolare evidenziando il migliore ed il peggiore tra questi. La fase successiva dell’esperimento consiste nell’esecuzione dell’algoritmo di generazione della cricca. L’algoritmo è stato eseguito due volte per ogni istanza, prima effettuando una ricerca non informata e poi una informata, utilizzando l’euristica descritta. In questo modo è stato possibile valutare gli insiemi di nodi generati dai due tipi di ricerca, confrontandoli con tutte le altre cricche esistenti nel grafo di connettività dell’istanza. Per ognuno dei due tipi di ricerca è stato calcolato, per ogni istanza, il grado di ottimalità raggiunto. Esso è un valore che mette in relazione il costo associato alla cricca (*ossia il costo della soluzione, che dipende dalla cricca scelta*) trovata dall’algoritmo di generazione, con la migliore e la peggiore cricca trovate tramite ricerca esaustiva. Questo indicatore è definito tramite la seguente formula:

dove *gcc* (*generated clique cost*) è il costo della cricca generata dall’algoritmo testato, mentre bcc (*best clique cost*) e wcc (*worst clique cost*) sono rispettivamente i costi della migliore e peggiore cricca trovata. Il grado di ottimalità sarà compreso nell’intervallo [0, 1] e assumerà valore 1 quando la cricca generata avrà un costo uguale a quella ottima, mentre varrà 0 quando essa avrà costo uguale alla peggiore trovata. Esiste un caso degenere, nel quale tutte le cricche trovate hanno il medesimo costo; in questo frangente non viene utilizzata la formula descritta (*in quanto porterebbe all’esecuzione di una divisione per 0*) ma viene assegnato un valore pari a 1, poiché se tutti gli insieme di nodi di *goal* esistenti portano al medesimo costo per una soluzione, qualunque venga selezionato costituisce di fatto una scelta ottima. Con questo metodo è stato possibile raccogliere dati significativi riguardanti la qualità delle soluzioni trovate tramite il processo definito in questo elaborato. Sono state inoltre ottenute informazioni relative ai tempi di esecuzione dell’algoritmo di generazione, che variano in base all’utilizzo dell’euristica.

Il secondo esperimento svolto ha l’obiettivo di determinare se sia ragionevole utilizzare l’algoritmo ungherese, i cui assegnamenti trovati sono sempre ottimi, per accoppiare ad ogni agente un nodo al quale dirigersi. Si è voluto verificare quanto la scelta dell’assegnamento influisse sul costo della soluzione e se fosse stato conveniente utilizzare invece un assegnamento casuale. Per raccogliere i dati richiesti, sono state risolte, come nel caso del primo esperimento, centinaia di istanze del problema. Per ognuna di esse, è stato prima determinato l’insieme di nodi di *goal* utilizzando l’algoritmo di generazione della cricca. Successivamente, sono stati generati due diversi assegnamenti: quello ottimo, trovato con l’algoritmo ungherese, ed uno casuale. È stato in seguito calcolato tramite CBS un insieme di percorsi per entrambe le versioni della soluzione formulate. In questo modo è stato possibile determinare il costo della soluzione e come esso varii, a parità di cricca scelta, utilizzando l’assegnamento ottimo ed uno generato casualmente. Si è inoltre monitorato il tempo di esecuzione dell’algoritmo ungherese, per poter determinare se esso fosse significativo, mentre per la generazione casuale esso è chiaramente trascurabile.

Come attestato in [[17](#riferimento_17)], sebbene CBS sia un algoritmo completo ed ottimo, non si hanno garanzie sul tempo effettivo di esecuzione. Nonostante esso sia solitamente in grado di produrre una soluzione in tempi mediamente brevi, possono presentarsi istanze particolarmente ardue per le quali sono necessari svariati minuti di esecuzione. Per poter svolgere gli esperimenti descritti, nei quali per ogni istanza di R-MAPF vanno risolte con CBS decine se non centinaia di istanze di MAPF, è stato imposto un limite di un minuto all’algoritmo, oltre il quale esso viene forzatamente interrotto. In questi casi, dove non è stato possibile ottenere il costo della soluzione per l’istanza, è stato utilizzato un *lower bound*, definito come la somma dei costi dei percorsi minimi per gli agenti, trovati grazie ad A\*, ignorando i possibili conflitti.

Un’altra questione meritevole di attenzione riguarda gli assegnamenti agente-*goal*. Come già esplicato nel Capitolo precedente, per poter determinare un assegnamento tramite l’algoritmo ungherese, si è scelto di utilizzare A\* per associare ad ogni coppia di posizioni (*agent start position*, *goal position*) un costo, *lower bound* del costo effettivo. Come conseguenza, può succedere che due assegnamenti considerati entrambi ottimi dall’algoritmo ungherese, abbiano poi costi differenti una volta che il problema venga risolto con CBS. Raramente può anche capitare che un assegnamento considerato peggiore secondo l’euristica, porti ad una soluzione con costo minore. Si è comunque riscontrato nei dati raccolti come tali variazioni di costo siano limitate sia nella frequenza di occorrenza che nelle dimensioni.

Si segnalano inoltre ulteriori specifiche relative ai due esperimenti svolti. Per entrambi si sono utilizzate mappe di forma quadrata di quattro diverse dimensioni: 8x8, 10x10, 12x12 e 15x15. Per ogni grandezza sono state testate istanze con una densità di ostacoli pari al 10%, al 20% e al 30%. Per tutte le istanze testate è stato scelto di utilizzare un numero standardizzato di agenti pari a 8. Questo poiché il numero è rappresentativo di istanze del problema di piccole-medie dimensioni, ma anche perché con una tale quantità di agenti è risultato fattibile generare tutte le cricche della medesima dimensione presenti in un’istanza in tempi ragionevoli, impresa rivelatasi assai più ardua con gruppi di agenti più numerosi. Per ogni istanza, il grafo di connettività ad essa associato è stato generato utilizzando come criterio di connettività il costo del percorso più breve tra due vertici. Due nodi A e B sono cioè connessi se il percorso privo di ostacoli più breve che li collega, calcolabile con A\*, ha costo minore o uguale ad un parametro *d*.

4.3 Analisi dei dati raccolti

Considerando innanzitutto il primo esperimento svolto, il primo dato utile ottenuto riguarda il grado di ottimalità delle cricche generate. Come era possibile immaginare, i dati suggeriscono che una ricerca informata sia preferibile rispetto ad una non informata. Nella Figura 11 viene mostrata la distribuzione statistica del valore del grado di ottimalità delle cricche generate. La differenza tra le due tipologie di ricerca è evidente: per quanto riguarda la generazione non informata, nella fetta centrale delle istanze risolte (*intervallo tra primo e terzo quartile, contenente il 50% dei valori osservati*) il valore è compreso tra poco più di 0 e circa 0,60, con media pari a 0,38; servendosi invece dell’euristica proposta ed eseguendo quindi una ricerca informata, la maggior parte delle istanze risolte presentano un grado di ottimalità superiore a 0,85, con media pari a 0,90. È quindi ragionevole assumere che l’utilizzo dell’euristica scelta comporti un miglioramento sostanziale della qualità delle soluzioni generate. Si può notare come per entrambi i tipi di ricerca esistano istanze per le quali il grado di ottimalità raggiunga il massimo ed il minimo valore possibile, con tuttavia una marcata differenza in termini di frequenza. Per la ricerca non informata, i valori di massimo e minimo giacciono entrambi entro i “baffi” del *box plot* e si trovano quindi ad una distanza inferiore a 1,5 IQR (*interquartile range, la distanza tra terzo e primo quartile*) in relazione a, rispettivamente, terzo e primo quartile. Il grafico evidenzia in particolare una notevole quantità di istanze per cui il valore è 0 o approssimabile ad esso. Per quanto riguarda invece la ricerca informata, ogni valore inferiore a 0.7 costituisce un *outlier* ed è cioè un valore che si allontana decisamente dalla distribuzione osservata, trovandosi oltre 1,5 IQR sotto il primo quartile. In conclusione, l’euristica proposta porta ad un miglioramento significativo delle prestazioni dell’algoritmo che, facendone uso, permette di ottenere soluzioni mediamente molto vicine all’ottimo, pur essendo subottimale. Sono presenti istanze in cui il grado di ottimalità risulta essere basso, ma ciò capita raramente.

Immagine che contiene testo, diagramma, schermata, Parallelo

Descrizione generata automaticamente

Figura 11: grado di ottimalità delle soluzioni trovate tramite generazione informata e non informata dell’insieme di nodi di *goal*. Le linee arancioni indicano i valori mediani mentre i triangoli verdi segnalano la media. I piccoli cerchi neri rappresentano gli *outlier*.

Avendo discusso della variazione in termini di qualità delle soluzioni trovate, è ora il momento di analizzare i dati riguardanti i tempi di esecuzione, che sono il secondo ambito di interesse del primo esperimento. Come è possibile vedere nella Figura 12, i tempi di esecuzione richiesti dalla generazione di un insieme di nodi di *goal* variano sensibilmente qualora la ricerca effettuata sia informata o meno. Mentre per la generazione non informata i tempi sono concentrati in un intervallo compreso tra circa 20 e 90 millisecondi (ms), per la ricerca informata essi sono fortemente concentrati tra 0 e 20 ms, con gran parte delle istanze terminate quasi istantaneamente, in un tempo approssimabile a 0 ms. I dati suggeriscono quindi che utilizzando l’euristica proposta e rendendo la ricerca informata, l’algoritmo di generazione della cricca richieda meno tempo per restituire un insieme di nodi. Questo risultato non è affatto sorprendente, è infatti ragionevole pensare che una ricerca informata possa giungere più velocemente alla generazione di una cricca, rispetto ad una non informata, nella quale spesso si dovrà effettuare *backtracking* per far uscire la ricerca da situazioni di stallo, derivanti dall’espansione di nodi effettuata senza alcun criterio. È infine doveroso segnalare che dal grafico mostrato sono stati rimossi gli *outlier*, sono infatti presenti alcune istanze per le quali sono stati necessari fino a 4 secondi per l’esecuzione dell’algoritmo testato. Tali istanze costituiscono un’anomalia e non sono state rappresentate in quanto statisticamente irrilevanti, oltre che per rendere il grafico più leggibile.

Immagine che contiene testo, diagramma, schermata, linea

Descrizione generata automaticamente

Figura 12: tempi di esecuzione richiesti dall’algoritmo di generazione dell’insieme di nodi di *goal* qualora la ricerca fosse non informata o informata. Le linee arancioni indicano i valori mediani.

I primi dati raccolti delineano dunque un quadro decisamente chiaro: l’utilizzo della ricerca informata per la generazione dell’insieme di nodi di *goal* è altamente preferibile ad una ricerca non informata. L’utilizzo dell’euristica proposta comporta un miglioramento significativo di entrambe le statistiche considerate, ossia il grado di ottimalità della soluzione e il tempo di esecuzione richiesto dall’algoritmo. Si è inoltre appurato come, facendo uso dell’euristica, l’algoritmo di generazione della cricca permetta di ottenere mediamente soluzioni molto vicine all’ottimo, richiedendo un tempo di esecuzione contenuto.

Sebbene l’obiettivo dichiarato per il primo esperimento sia già stato raggiunto, è comunque possibile utilizzare i dati raccolti per generare ulteriori grafici e poter in questo modo inferire nuove informazioni sull’algoritmo sviluppato. È ad esempio interessante osservare il comportamento dell’algoritmo al variare di alcune condizioni del problema. Una di queste variabili è il numero di cricche della dimensione cercata, esistenti all’interno del grafo di connettività di un’istanza. Tale numero è strettamente legato ad altri parametri del problema, quali la dimensione del grafo primario, la densità degli ostacoli presenti e la dimensione della cricca voluta (*corrispondente al numero di agenti presenti*). Logicamente, diminuendo la dimensione della cricca ricercata, mantenendo invariato il grafo di connettività, ne esisteranno in numero maggiore, mentre aumentando il numero di ostacoli presenti nell’ambiente, il numero di cricche diminuirà, in quanto ci saranno meno vertici liberi in cui poter far incontrare gli agenti. Tali considerazioni sono piuttosto banali; può invece essere interessante mettere in relazione il numero di cricche della dimensione ricercata presenti in un’istanza con le prestazioni dell’algoritmo di generazione. Nella Figura 13 si può notare come per la ricerca non informata tali valori siano debolmente correlati. Sono assenti istanze con ottimalità superiore a 0,8 quando il numero di cricche è maggiore di 75, oltre a questo però, la distribuzione dei valori di ottimalità sembra non essere collegata al numero di cricche presenti. Nella Figura 14, si può invece individuare una correlazione decisamente più forte tra i due fattori, quando la ricerca svolta è informata. Il grafico evidenzia come la generazione informata abbia maggiori probabilità di produrre una cricca che si avvicini a quella ottima all’aumentare del numero di cricche esistenti. Si può infatti constatare come per istanze con grafi di connettività contenenti 75 o più cricche della dimensione richiesta, il grado di ottimalità non scenda mai sotto 0,8. Se invece il numero di cricche scende sotto 25, si incontrano le poche istanze per le quali il grado di ottimalità è inferiore a 0,6. Si può quindi affermare che la ricerca informata lavori decisamente meglio quando il numero di cricche esistenti è maggiore e che, al contrario, in tali frangenti la ricerca non informata, che già raramente porta a soluzioni di qualità, abbia meno probabilità di successo. Questo fenomeno può essere spiegato nel seguente modo: poiché la bontà di una cricca trovata da una ricerca non informata dipende da fattori aleatori, se esistono poche cricche tra cui scegliere è più probabile che ne venga selezionata una delle migliori tra quelle. Dove invece sono presenti molte decine o addirittura più di un centinaio di insiemi di nodi tra cui scegliere, la probabilità che ciò accada si abbassa e viene favorita la ricerca informata, che sfrutta la conoscenza del problema.

Immagine che contiene testo, schermata, diagramma, Carattere

Descrizione generata automaticamente

Figura 13: grado di ottimalità delle soluzioni trovate tramite generazione non informata dell’insieme di nodi di *goal*, rapportato al numero di cricche della dimensione ricercata presenti nell’istanza. Ogni punto blu rappresenta un’istanza diversa.

Immagine che contiene testo, schermata, diagramma

Descrizione generata automaticamente

Figura 14: grado di ottimalità delle soluzioni trovate tramite generazione informata dell’insieme di nodi di *goal*, rapportato al numero di cricche della dimensione ricercata presenti nell’istanza. Ogni punto blu rappresenta un’istanza diversa.

Oltre al numero di cricche presenti, è possibile considerare altri aspetti variabili delle istanze testate; uno di questi è la grandezza del grafo dell’istanza utilizzato per rappresentare l’ambiente di movimento degli agenti. Aggregando diversamente gli stessi dati raccolti per il primo esperimento, dividendoli in base alle dimensioni dell’istanza analizzata, è possibile studiare il comportamento dell’algoritmo di generazione in presenza di mappe più grandi. Sebbene le dimensioni utilizzate negli esperimenti svolti siano contenute e adatte a rappresentare istanze medio-piccole del problema, la differenza tra una mappa 8x8 ed una 15x15 è comunque notevole in quanto, ipotizzando per entrambe l’assenza di ostacoli, si passa da 64 vertici a 225, un aumento del 350%. Nella Figura 15 è rappresentata la distribuzione del valore di ottimalità delle cricche generate tramite ricerca non informata. Sebbene il calo non sia drastico, si può comunque osservare un andamento negativo, per il quale al crescere delle dimensioni dell’istanza, l’ottimalità dell’insieme di nodi generato tende a diminuire. Nella Figura 16 sono invece rappresentati i valori riguardanti la generazione informata. Si può notare come in questo caso la distribuzione dei valori sia fondamentalmente la stessa per le quattro categorie, con valori di media e mediano quadi identici, così come per i quartili. La differenza più notevole riguarda le istanze dalla mappa più grande, 15x15, dove, a differenza delle altre tre categorie, non compaiono valori di ottimalità inferiori a 0,5. È anche possibile osservare un lieve aumento della media, derivante da tale assenza. Questi dati vanno a rafforzare l’ipotesi formulata precedentemente, secondo cui in presenza di un numero ridotto di cricche la ricerca non informata ha maggiori probabilità di successo, mentre all’aumentare di queste, il divario tra generazione informata e non diventa ancora più evidente. A parità di densità di ostacoli infatti, più una mappa è grande e più cricche della dimensione cercata esistono al suo interno (*ciò è vero in questo particolare contesto, con il criterio di connettività usato per questi esperimenti*). Poiché per ogni categoria esaminata sono presenti istanze con diverse densità di ostacoli, equamente distribuite, l’aumento delle cricche al crescere della mappa è riscontrabile nei grafici.

Immagine che contiene testo, diagramma, Piano, Disegno tecnico

Descrizione generata automaticamente

Figura 15: grado di ottimalità delle soluzioni trovate tramite generazione non informata dell’insieme di nodi di *goal*, rapportato alle dimensioni del grafo dell’istanza. Le linee arancioni indicano i valori mediani mentre i triangoli verdi segnalano la media.

Immagine che contiene testo, schermata, diagramma, Piano

Descrizione generata automaticamente

Figura 16: grado di ottimalità delle soluzioni trovate tramite generazione informata dell’insieme di nodi di *goal*, rapportato alle dimensioni del grafo dell’istanza. Le linee arancioni indicano i valori mediani mentre i triangoli verdi segnalano la media.

Nelle Figure 17 e 18 si può osservare come una diversa grandezza del grafo influenzi anche il tempo di esecuzione richiesto dall’algoritmo, rispettivamente utilizzando una ricerca non informata e informata. In entrambi i grafici sono stati rimossi i valori *outlier*, come già effettuato precedentemente. Per quanto riguarda la ricerca non informata, la distribuzione dei valori è piuttosto incoerente: essa propende verso tempi più lunghi all’aumentare delle dimensioni del grafo, ma poi si sposta nuovamente verso tempi minori per le istanze di grandezza 15x15. Osservando i dati riguardandi la ricerca informata, si può notare un andamento similare. Risulta comunque particolarmente evidente come una mappa di dimensioni minori comporti tempi di esecuzione ridotti. Sebbene per tutte le dimensioni esaminate il valore mediano sia 0 ms, per le istanze di grandezza 8x8 il tempo richiesto dall’algoritmo è praticamente sempre trascurabile, mentre per mappe più grandi a volte viene richiesto qualche frazione di secondo in più. I risultati non sono sorprendenti: poiché il grafo di connettività di un’istanza di R-MAPF contiene gli stessi vertici del grado primario, per una mappa di dimensioni maggiori saranno presenti più vertici. Di conseguenza la ricerca su albero effettuata dall’algoritmo di generazione della cricca avrà potenzialmente più nodi da esplorare e potrà necessitare di un tempo maggiore.

Immagine che contiene testo, diagramma, Disegno tecnico, Piano

Descrizione generata automaticamente

Figura 17: tempo di esecuzione richiesto dall’algoritmo di generazione non informata dell’insieme di nodi di *goal*, rapportato alle dimensioni del grafo dell’istanza. Le linee arancioni indicano i valori mediani.

Immagine che contiene testo, diagramma, schermata, linea

Descrizione generata automaticamente

Figura 18: tempo di esecuzione richiesto dall’algoritmo di generazione informata dell’insieme di nodi di *goal*, rapportato alle dimensioni del grafo dell’istanza. Le linee arancioni indicano i valori mediani.

Un’altra variabile del problema che può essere analizzata è la densità di ostacoli presenti nel grafo di un’istanza. Tali ostacoli, che nell’ambiente dove si muovono gli agenti possono rappresentare pareti, mobili ed altri elementi solidi, nel grafo di un’istanza, sia primario che di connettività, possono essere immaginati come *vertici mancanti*, porzioni vuote del grafo. La navigazione degli agenti in un ambiente diventa più complessa all’aumentare degli ostacoli, in quanto calcolare i percorsi per farli muovere da un punto ad un altro diviene più difficile, dovendo passare intorno a tali punti che non possono essere attraversati. Possono in particolare crearsi dei *bottleneck*, ossia dei *colli di bottiglia,*passaggi che connettono due sezioni del grafo separate da ostacoli, dove, se più agenti devono passare simultaneamente, può diventare arduo per CBS calcolare percorsi privi di conflitti che lo permettano. In Figura 19 è rappresentato il legame tra la densità di ostacoli presenti e l’ottimalità delle cricche generate tramite ricerca non informata. All’aumentare degli ostacoli le prestazioni dell’algoritmo sprovvisto di euristica non peggiorano, la curva di distribuzione dei valori tende piuttosto ad appianarsi, comprendendo più frequentemente sia valori più bassi che più alti. Si noti ad esempio come con una densità del 10% l’ottimo non venga mai raggiunto, mentre con il 20% e il 30% sì. Questo grafico riafferma nuovamente quanto discusso precedentemente riguardo la relazione tra quantità di cricche della dimensione cercata presenti e ottimalità delle soluzioni. Con una densità maggiore di ostacoli ed il criterio di connettività usato per gli esperimenti, il numero di cricche sarà minore e la ricerca non informata avrà più probabilità di imbattersi casualmente in un insieme di nodi di *goal* che porti ad una soluzione dal costo contenuto. Sempre a conferma di questa teoria, si noti come in Figura 20, dove il grafico rappresenta le stesse informazioni ma riguardanti l’uso della generazione informata, la distribuzione dei valori di ottimalità si sposti verso l’estremo inferiore all’aumentare degli ostacoli. Questo grafico rivela inoltre come per tutte le istanze con ottimalità raggiunta inferiore a 0,7 la densità di ostacoli sia 20% o superiore, mentre per valori inferiori a 0,5 la densità di ostacoli sia sempre 30%.

Immagine che contiene testo, diagramma, linea, Parallelo

Descrizione generata automaticamente

Figura 19: grado di ottimalità delle soluzioni trovate tramite generazione non informata dell’insieme di nodi di *goal*, rapportato alla densità di ostacoli presenti nell’istanza. Le linee arancioni indicano i valori mediani mentre i triangoli verdi segnalano la media.

Immagine che contiene testo, diagramma, schermata, Parallelo

Descrizione generata automaticamente

Figura 20: grado di ottimalità delle soluzioni trovate tramite generazione informata dell’insieme di nodi di *goal*, rapportato alla densità di ostacoli presenti nell’istanza. Le linee arancioni indicano i valori mediani mentre i triangoli verdi segnalano la media.

Nelle Figure 21 e 22 viene infine messa in relazione la densità di ostacoli presenti con il tempo di esecuzione richiesto dall’algoritmo, rispettivamente con ricerca non informata e informata. Si può notare come, per entrambi i tipi di ricerca, l’aumentare degli ostacoli presenti comporti una probabilità maggiore che l’algoritmo richieda tempi più lunghi. Quando la generazione fa uso dell’euristica proposta l’incremento nel tempo richiesto è particolarmente netto. Ciò accade poiché con una maggiore frequenza di ostacoli nel grafo, il numero di connessioni tra vertici all’interno del grafo di connettività di un’istanza tende a diminuire. È di conseguenza più probabile che, durante la ricerca su albero, l’algoritmo si ritrovi in un punto di stallo, nel quale non è possibile aggiungere vertici alla cricca generata fino a quel punto ed è quindi necessario effettuare *backtracking*.

Immagine che contiene testo, diagramma, schermata, Disegno tecnico

Descrizione generata automaticamente

Figura 21: tempo di esecuzione richiesto dall’algoritmo di generazione non informata dell’insieme di nodi di *goal*, rapportato alla densità di ostacoli presenti nell’istanza. Le linee arancioni indicano i valori mediani.

Immagine che contiene testo, schermata, diagramma, linea

Descrizione generata automaticamente

Figura 22: tempo di esecuzione richiesto dall’algoritmo di generazione informata dell’insieme di nodi di *goal*, rapportato alla densità di ostacoli presenti nell’istanza. Le linee arancioni indicano i valori mediani.

In conclusione, si è evidenziato come le dimensioni della mappa e la densità di ostacoli in essa presenti siano fattori che possano influenzare la difficoltà di un’istanza di R-MAPF. Aumentare tali parametri porta infatti statisticamente a valori di ottimalità minori e tempi di esecuzione più lunghi. Tra i due fattori, la densità di ostacoli è risultata quello più determinante. Rimane comunque evidente la superiorità della generazione facente uso dell’euristica proposta rispetto alla ricerca non informata.

Verranno ora analizzati i dati riguardanti il secondo esperimento svolto. Per poter valutare l’impatto dell’utilizzo dell’algoritmo ungherese sulla qualità delle soluzioni prodotte, è stata considerata, per ogni istanza testata, la differenza tra il costo della soluzione con l’assegnamento trovato dall’algoritmo ungherese e il costo della soluzione per la quale è stato scelto un assegnamento casuale. Nella Figura 23, nella quale si considera il costo secondo l’euristica, lo stesso utilizzato dall’algoritmo ungherese, viene riportata la distribuzione di tale valore sotto forma di percentuale. Naturalmente la differenza percentuale è sempre positiva, in quanto l’algoritmo ungherese è ottimo e di conseguenza non può esistere un assegnamento casuale che abbia costo inferiore ad uno trovato con tale algoritmo. I dati mostrano un miglioramento significativo in termini di costo delle soluzioni trovate; per gran parte delle istanze esso è compreso tra circa il 15% ed il 30%, ma non è raro che esso arrivi anche oltre il 30%. Osservando la Figura 24, nella quale viene invece considerato il costo reale della soluzione, calcolato tramite CBS, si può constatare come la distribuzione statistica subisca alcuni cambiamenti. Per buona parte delle istanze, il miglioramento registrato si attesta sempre tra circa il 15% e il 30%, è tuttavia più comune che esso arrivi anche al 40% o addirittura a 50%. Si può anche notare come, per una ridotta porzione di istanze, l’assegnamento trovato dall’algoritmo ungherese porti ad una soluzione dal costo lievemente maggiore rispetto a quella facente uso dell’assegnamento casuale. Sono infatti presenti valori percentuali compresi tra lo 0 e il -10%. Ciò accade per il fenomeno descritto in precedenza, per il quale un assegnamento uguale o addirittura peggiore a quello ottimo generato usando come base un’euristica, può poi in rari casi rivelarsi migliore una volta che il costo reale della soluzione viene calcolato.

Immagine che contiene testo, schermata, diagramma, linea

Descrizione generata automaticamente

Figura 23: miglioramento in termini di costo della soluzione derivante dall’utilizzo dell’algoritmo ungherese, rispetto ad un assegnamento generato casualmente. Viene considerato il costo calcolato tramite un euristica (A\*). La linea arancione indica il valore mediano.

Immagine che contiene testo, diagramma, schermata, Rettangolo

Descrizione generata automaticamente

Figura 24: miglioramento in termini di costo della soluzione derivante dall’utilizzo dell’algoritmo ungherese, rispetto ad un assegnamento generato casualmente. Viene considerato il costo reale, calcolato grazie a CBS. La linea arancione indica il valore mediano.

Considerando i dati raccolti, risulta evidente il beneficio derivante dall’uso dell’algoritmo ungherese, per la determinazione dell’assegnamento agente-*goal* in un’istanza di R-MAPF. A fronte di un così notevole miglioramento in termini di costi, servirsi di un assegnamento casuale potrebbe essere ragionevole solamente se ciò portasse ad un risparmio nei tempi di esecuzione richiesti. Come viene però mostrato nella Figura 25, l’algoritmo richiede tempi molto contenuti, che non eccedono mai i 30 ms e nella maggior parte dei casi sono addirittura inferiori a 15 ms. Di conseguenza, sulla base dei dati raccolti, si può affermare che per la generazione dell’assegnamento l’algoritmo ungherese sia la scelta più ragionevole, mentre l’utilizzo di un assegnamento casuale non è affatto conveniente, non portando a risparmi significativi di tempo e abbassando sistematicamente la qualità delle soluzioni.

Immagine che contiene testo, schermata, diagramma, Rettangolo

Descrizione generata automaticamente

Figura 25: tempi di esecuzione richiesti dall’algoritmo ungherese per determinare un assegnamento agente-*goal* per tutti gli agenti presenti in un’istanza di R-MAPF.

Capitolo 5

Conclusioni e sviluppi futuri

In questa tesi è stato definito il problema R-MAPF, per il quale è stato proposto un metodo di risoluzione, che comporta la riduzione di un’istanza del problema ad un caso di MAPF e l’utilizzo di CBS per la generazione dei percorsi per gli agenti. Sono stati effettuati degli esperimenti per verificare l’efficacia e l’efficienza del processo presentato e i dati raccolti sono stati analizzati e commentati mostrando come esso, pur essendo subottimale, offra mediamente tempi di esecuzione limitati ed un alto grado di ottimalità per le soluzioni trovate.

In futuro il lavoro esposto potrebbe essere ampliato, in diversi aspetti. La definizione del problema potrebbe essere estesa, per includere aspetti propri delle varianti di MAPF menzionate nel Capitolo 2. Si potrebbe ad esempio fare in modo che con R-MAPF si possano rappresentare scenari in cui gli agenti abbiano forme e dimensioni variabili o i cui piani di movimento debbano soddisfare regole di robustezza o di formazione. Concentrandosi invece sul metodo di risoluzione, si potrebbero sviluppare tecniche per aumentarne ulteriormente le prestazioni, in termini di tempo di esecuzione e ottimalità raggiunta. Si potrebbe, ad esempio, progettare una procedura che, lavorando insieme a CBS, possa riorganizzare in tempo reale l’assegnamento degli agenti ai *goal*, nel caso in cui durante l’esecuzione dei percorsi emerga la possibilità di usarne uno migliore.

Riferimenti bibliografici

[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), vol. 10*,2019. DOI: <https://doi.org/10.1609/socs.v10i1.18510>

[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>

[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: <https://doi.org/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: <https://doi.org/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>

[17] G. Sharon, R. Stern, A. Felner, N. R. Sturtevant, “Conflict-based search for optimal multi-agent pathfinding” *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2012. DOI: <https://doi.org/10.1016/j.artint.2014.11.006>

[18] P. E. Hart, N. J. Nilsson, B. Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths” *IEEE Transactions on Systems Science and Cybernetics vol. 4*, *no. 2*, pp. 100-107, 1968. DOI: <https://doi.org/10.1109/TSSC.1968.300136>

[19] D. Yao, Z. Zhao, W. Ng, “Efficient processing of optimal meeting point queries in Euclidean space and road networks” *Knowl Inf Syst 42*, pp. 319–351, 2015. DOI: <https://doi.org/10.1007/s10115-013-0686-y>

[20] D. Atzmon, A. Felner, J. Li, S. Shperberg, N. Sturtevant, S. Koenig, “Conflict-tolerant and conflict-free multi-agent meeting” *Artificial Intelligence vol. 322*, 2023. DOI: <https://doi.org/10.1016/j.artint.2023.103950>

[21] S. Kloder e S. Hutchinson, “Path planning for permutation-invariant multirobot formations” *IEEE Transactions on Robotics, vol. 22, no. 4*, pp. 650-665, 2006. DOI: <https://doi.org/10.1109/TRO.2006.878952>

[22] J. Yu e S. M. LaValle, “Multi-agent Path Planning and Network Flow” *Multi-agent Path Planning and Network Flow. In: Frazzoli, E., Lozano-Perez, T., Roy, N., Rus, D. (eds) Algorithmic Foundations of Robotics X. Springer Tracts in Advanced Robotics, vol 86.* 2013. DOI: <https://doi.org/10.1007/978-3-642-36279-8_10>

[23] H. W. Kuhn, "The Hungarian method for the assignment problem." *Naval research logistics quarterly vol. 2 issue 1‐2*, pp. 83-97, 1955. DOI: <https://doi.org/10.1002/nav.3800020109>

1. fonte: J. Li. Indirizzo: <https://jiaoyangli.me/research/mapf/> [↑](#footnote-ref-1)
2. 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-2)
3. È possibile trovare la libreria utilizzata presso la *repository* <https://github.com/SvetaLadigin/robotics_mini_project> [↑](#footnote-ref-3)
4. <https://visualstudio.microsoft.com/> [↑](#footnote-ref-4)
5. Si tratta della libreria Munkres, la cui documentazione è consultabile presso l’indirizzo <https://software.clapper.org/munkres/> [↑](#footnote-ref-5)