Relazione Intelligenza Artificiale

Università degli studi di Brescia

Thomas Causetti 731077
Gabriele Ceresara 729324
Jacopo Tedeschi

A.A. 2025-2026

Gennaio-Febbraio 2026

Indice

0.1 Introduzione

Questo progetto si basa sull'implementazione di un sistema di Planning per il problema che è stato presentato nella competizione LORR24 (League of Robot Runners 2024) url: `https://2024.leagueofrobotrunners.org/`. Questa competizione è dedicata al problema del Multi-Agent Path Finding (MAPF) in ambienti dinamici. In questo contesto, più agenti robotici devono navigare in una griglia bidimensionale, evitando collisioni sia di vertice che di spigolo, mentre completano una sequenza di compiti assegnati dinamicamente. Il sistema deve gestire vincoli temporali stringenti, con limiti di tempo per la pianificazione in ogni timestep, e ottimizzare metriche come il tempo di completamento dei compiti e l'efficienza complessiva.

L'approccio implementato in questo progetto è un planner ibrido che combina quattro algoritmi principali:

- **Large Neighborhood Search (LNS)**: ottimizzazione delle traiettorie a lungo termine.
- **Lazy Constraints Addition Search for MAPF (LaCAM)**: Utilizzato come preprocessing per LNS.
- **Priority Inheritance with Backtracking (PIBT)**: risoluzione rapida dei conflitti a breve termine. (Solo come fallback in caso di fallimento di LNS)
- **Rolling Horizon Collision Resolution (RHCR)**: gestione dinamica dei conflitti in tempo reale. (Solo come fallback in caso di fallimento di LNS)

Utilizzando diversi algoritmi in combinazione, l'implementazione mira ad otternere i vantaggi di ogni metodo, migliorando l'efficacia complessiva del sistema di pianificazione.

Nei capitoli successivi, i 4 diversi algoritmi saranno descritti in dettaglio, insieme alle scelte implementative e alle strategie adottate per affrontare le sfide specifiche del problema MAPF in ambienti dinamici. Nel seguito della relazione, descriveremo l'architettura del sistema, i dettagli implementativi, i risultati sperimentali e le valutazioni ottenute nella competizione.

0.2 Scelta degli algoritmi

Gli algoritmi scelti per l'implementazione del planner ibrido sono stati selezionati effettuando una valutazione delle varie implementazioni di diversi team della competizione sia nell'anno corrente che in quelli passati. Studiando le varie soluzioni proposte, abbiamo notato come l'approccio ibrido che combina più algoritmi abbia portato a risultati migliori rispetto all'utilizzo di un singolo algoritmo. Infine per inplementare il planner abbiamo cercato di capire come implementare gli algoritmi scelti con paper pubblicati in letteratura e con le implementazioni open source disponibili.

0.3 Preprocessing

La competizione fornisce un tempo per il preprocessing, che il nostro planner utilizza per precalcolare le euristiche pesate tra tutte le coppie di posizioni sulla mappa. Questo processo sfrutta una ricerca A* a piu livelli che inizializza rapidamente tutte le euristiche

con la distanza di Manhattan, per poi raffinarle progressivamente utilizzando il tempo disponibile. Il calcolo avviene in tre fasi: prima l'inizializzazione con distanza di Manhattan (molto veloce), poi un raffinamento A* senza considerare le rotazioni (più veloce), ed infine un raffinamento completo con A* che considera anche le orientazioni (più accurato). Il preprocessing è parallelizzato con OpenMP per sfruttare al meglio i core disponibili.

Il calcolo è stato fatto a più livelli imitando un pò l'approccio Anytime, perchè dato che il tempo di preprocessing è limitato, e il software successivo necessita di queste euristiche, nei problemi più complessi, il programma non riesce a completare il calcolo di tutte le euristiche, ma riesce comunque a fornire al software successivo delle euristiche iniziali, raffinate per tutto il tempo disponibile. In questo modo, si garantisce la presenza di un euristica e si rende impossibile il timeout del preprocessing, anche in scenari complessi.

Per scenari semplici l'overhead fornito non è notabile, dato che il calcolo è delle euristiche di primo e secondo livello è molto veloce.

0.4 Large Neighborhood Search (LNS)

0.4.1 Principio di funzionamento

Il principio di funzionamento del Large Neighborhood Search (LNS) applicato al Multi-Agent Pathfinding (MAPF) si basa su un approccio metaeuristico che combina distruzione e riparazione iterativa delle soluzioni. L'obiettivo è quello di ottimizzare i percorsi di più agenti in un ambiente condiviso, minimizzando i conflitti e i costi complessivi.

L'algoritmo LNS inizia con una soluzione iniziale, che nel caso di questo progetto viene calcolata utilizzando LaCAM. Successivamente, una parte della soluzione viene distrutta selezionando un sottoinsieme di agenti e rimuovendo i loro percorsi. Questa selezione può essere effettuata utilizzando diverse euristiche, come la scelta casuale, la selezione degli agenti più ritardati o quelli che si trovano in posizioni di conflitto.

Una volta distrutta parzialmente la soluzione, l'algoritmo tenta di ripararla ricalcolando i percorsi per gli agenti selezionati. Questo processo di riparazione utilizza un planner, come il Time-Space A* con consapevolezza delle orientazioni, che considera i vincoli temporali e spaziali per evitare conflitti. L'algoritmo valuta la nuova soluzione e decide se accettarla in base a criteri di ottimalità o accettazione probabilistica.

L'implementazione del LNS è strutturata in diverse componenti principali:

- **PathT e PathTableWC**: Queste strutture gestiscono l'occupazione spazio-temporale e il rilevamento dei conflitti. Consentono di registrare e annullare i percorsi degli agenti, verificare blocchi e calcolare conflitti futuri.
- **ConstraintTable**: Questa classe gestisce i vincoli rigidi e morbidi, come i blocchi di movimento e le penalità, per garantire che i percorsi rispettino i vincoli definiti.
- **ReservationTable**: Utilizzata per calcolare intervalli di tempo sicuri per gli agenti, considerando i vincoli e i conflitti esistenti.
- **TimeSpaceAStarPlanner**: Un planner avanzato che utilizza una variante del classico A* per trovare percorsi ottimali minimizzando i conflitti. Tiene conto delle orientazioni degli agenti e dei vincoli temporali.

- **Parallel LNS**: Una versione parallela dell'algoritmo che sfrutta il calcolo multi-thread per migliorare le prestazioni, suddividendo il problema in sottoproblemi gestiti indipendentemente.

L'algoritmo è stato progettato per essere modulare e flessibile, consentendo di adattare le euristiche di distruzione e riparazione in base alle esigenze specifiche del problema. L'uso di strutture dati efficienti, come le heap per le code di priorità e le tabelle hash per il rilevamento dei conflitti, garantisce prestazioni elevate anche in scenari complessi con molti agenti.

0.4.2 Ottimmizzazione

Per ottimizzare ulteriormente le prestazioni del LNS si è fatto profiling del codice con perf, per idendificare eventuali colli di bottiglia. L'heap utilizzata per la coda di priorità è stata modificata da una versione 2 heap iniziale a una 4 heap, che ha portato a un miglioramento significativo delle prestazioni. Inoltre, sono state implementate versioni parallele del LNS, sfruttando il calcolo multi-thread per gestire sottoproblemi in modo indipendente, riducendo così i tempi di esecuzione complessivi.

Inoltre si è modificato rispetto a una versione iniziale di LNS la gestione del tempo massimo di esecuzione. Dato che ogni timestep ha un limite di tempo di 1 secondo, inizialmente si impostava un limite di 0.9 secondi per l'esecuzione del LNS. Questa versione funzionava bene per istanze più semplici, ma per istanze più complesse, si notava che l'algoritmo andava in timeout per le prime n iterazioni, stabilizzandosi successivamente. Successivamente, si è capito che per istanze più complesse, il tempo di esecuzione dello scheduler, soprattutto nella fase iniziale, richiedeva più tempo. Infine il problema è stato risolto modificando il funzionamento del calcolo del tempo massimo, non più basato su un tempo fisso, ma calcolato dinamicamente in base al tempo impiegato dallo scheduler. In questo modo, se lo scheduler utilizza 0.4 secondi, ad esempio, viene utilizzato il tempo rimantente, ovvero 0.6 secondi, come valore massimo per il calcolo del tempo di esecuzione del LNS. Il tempo di esecuzione verra calcolato utilizzando il cutoff time value presente in config.hpp, che indica quanto tempo, del tempo rimanete, viene utilizzato per esecuzione del LNS. (Il cutoff time è settato a circa 0.9 secondi, dato che si vuole lasciare un margine di tempo per eventuali operazioni di post processing e anche perchè il controllo che ferma l'esecuzione viene lanciato solo ogni n iterazioni altrimenti diventerebbe un collo di bottiglia).

0.5 Lazy Constraints Addition Search for MAPF (LaCAM)

0.5.1 Principio di funzionamento

Il principio di funzionamento di LaCAM (Lazy Constraints Addition Search for MAPF) si basa sull'idea di risolvere il problema del Multi-Agent Path Finding (MAPF) in modo incrementale, aggiungendo vincoli in modo "lazy" solo quando necessario. L'approccio è progettato per gestire efficacemente i conflitti tra agenti, evitando di dover considerare tutte le possibili combinazioni di traiettorie fin dall'inizio. Nel caso di questo progetto, LaCAM è stato utilizzato come fase di preprocessing per ottimizzare le traiettorie a lungo termine prima di applicare l'algoritmo LNS (Large Neighborhood Search) per ulteriori ottimizzazioni.

0.6 Priority Inheritance with Backtracking (PIBT)

0.6.1 Principio di funzionamento

0.7 Rolling Horizon Collision Resolution (RHCR)

0.7.1 Principio di funzionamento

0.8 Implementazione e Launcher

L'implementazione del planner è stata realizzata in C++, per garantire prestazioni migliori rispetto a codice Python.

Per semplificare l'esecuzione del codice, è stato sviluppato un launcher in Python che si occupa di gestire l'esecuzione del planner. Il launcher consente di installare le dipendenze necessarie e il visualizzatore PlanViz. Inoltre, permette di complilare ed eseguire il codice direttamente da GUI. Dall'interfaccia grafica è anche possibile selezionare esecuzioni precedenti e visualizzare il plan su PlanViz.

0.9 Risultati

Dato che la competizione è terminata non è stato possibile confrontare i risultati ottenuti con quelli degli altri team, perchè anche se pubblicati, le nostre macchine non possono avere le stesse prestazioni dei server forniti dalla competizione. Quindi, un confronto diretto risulterebbe poco significativo.

I risultati sono stati confrontati con il solver che viene fornito dalla competizione.

Il software fornito è un planner basato su PIBT e RHCR, ed è implementato molto bene.

Il nostro planner è risultato essere in grado di risolvere tutti i problemi con un numero di task terminati maggiore rispetto al solver fornito, come visibile nel seguente grafico, che mostra il numero di task completati per ogni problema, confrontando il nostro planner con quello fornito dalla competizione.

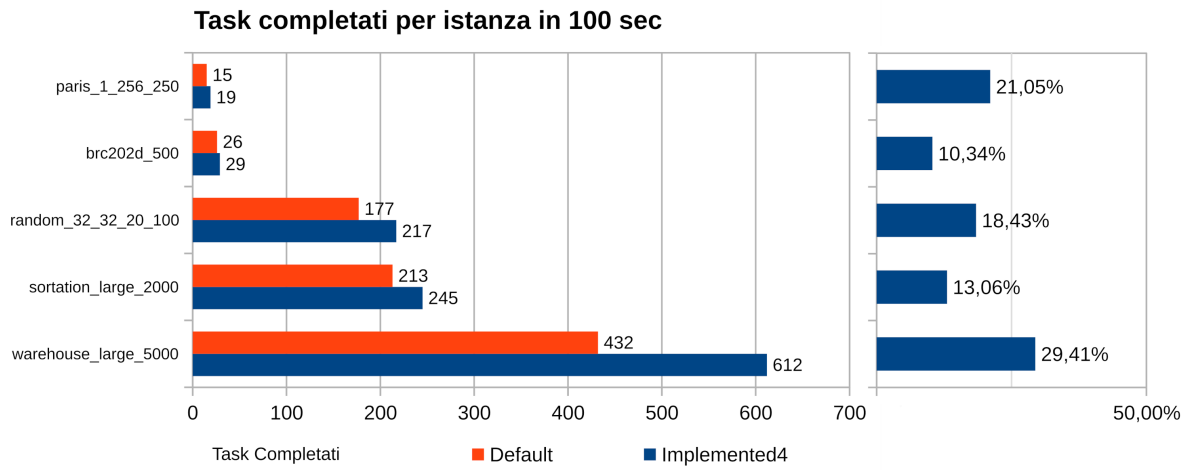

Figura 1: Confronto del numero di task completati in 100 secondi tra il planner implementato (Implemented4) e il solver di default fornito dalla competizione, con a destra i margini di vantaggio percentuali del nostro planner in ogni problema.

0.10 Conclusioni