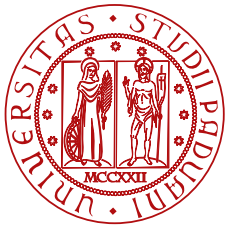

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in Ingegneria Informatica

Risoluzione di problemi di AI Planning tramite un approccio di tipo MIP

Relatore

Domenico Salvagnin

Università di Padova

Laureando

Gabriel Taormina

1219622

Anno Accademico 2021/2022

Data di laurea: 25/11/2022

Abstract

L'attività di AI Planning permette di raggiungere un obiettivo non inizialmente verificato partendo da uno stato iniziale tramite una sequenza di azioni per spostarsi tra gli stati con costo ed effetto. Si utilizzano strategie di ricerca euristica con risultati molto efficienti, è possibile tuttavia un approccio differente tramite l'utilizzo della programmazione lineare intera. Nello specifico di questo studio vengono affrontati due esperimenti per valutare l'efficacia dell'approccio MIP, tenendo conto del fatto che l'uso di un opportuno euristico avrà comunque risultati migliori

Indice

1

Introduzione

Il capitolo si occupa di fornire gli strumenti necessari per interpretare gli esperimenti effettuati, le strategie, i metodi e il tipo di modello utilizzato per la risoluzione delle istanze

1.1 AI Planning

AI planning è una branca dell'intelligenza artificiale che si occupa della risoluzione di problemi con molte applicazioni nella pianificazione dell'industria, robotica, guida autonoma e scheduling. I problemi e l'approccio utilizzato hanno una caratteristica ben definita. Lo scopo è quello di trovare una linea d'azione procedurale affinché un sistema raggiunga un obiettivo non verificato inizialmente, partendo da uno stato iniziale, attraverso una sequenza di azioni, aventi un determinato costo ed effetto per spostarsi tra i vari stati; ottimizzando nel complesso le prestazioni cercando la soluzione dal costo minore per raggiungere lo stato finale detto $goal$. In modo da sintetizzare dinamicamente il piano di azioni per raggiungere l'obiettivo a partire dallo stato iniziale. I problemi sono definiti da un insieme di azioni e di variabili, entrambe aventi cardinalità finita. Le azioni sono definite da tuple aventi un costo, una condizione per essere eseguita e un effetto, lo spazio degli stati è descritto in modo esplicito [7]. I problemi di AI Planning sono spesso scritti con la grammatica Planning Domain Definition Language (PDDL)[1]. PDDL cerca di standardizzare i linguaggi fino a quel momento usati nel campo di AI planning. È un linguaggio centrato sulle azioni con una sintassi standard per esprimerle, tuttavia non ha una semplice potenza espressiva. Nel complesso avrò bisogno di un insieme

di descrizioni di operatori/azioni, una descrizione dello stato iniziale e una descrizione dello stato finale.

1.2 A*

Una buona rappresentazione e approccio a problemi di AI planning vede l'uso di grafi, con particolare attenzione nel caso della risoluzione ad algoritmi per l'attraversamento di un grafo. Attualmente i migliori risultati sono stati ottenuti tramite una strategia di ricerca euristica, in particolar modo tramite l'utilizzo dell'algoritmo A* [16] grazie alle sue elevate prestazioni. A* è un algoritmo per l'attraversamento di grafi pesati tramite il percorso minimo basato su una funzione euristica ammissibile. Una funzione euristica è detta ammissibile se non sbagli mai per eccesso la stima del costo per arrivare al suo obiettivo. L'algoritmo valuta ciascun nodo con una funzione $f(n) = h(n) + g(n)$ dove:

- $h(n)$**:** costo percorso fino al nodo ispezionato
- $g(n)$**:** funzione euristica

Invece di considerare sola la distanza dall'obiettivo considera anche il costo nel raggiungere il nodo N dalla radice. Si espandono i nodi in ordine crescente di $f(n)$ e si sceglie come nodo da espandere quello con $f(n)$ minore. A* è un caso particolare della funzione di valutazione Best First, funzione che calcola un numero che rappresenta la desiderabilità relativa all'espansione del nodo. A* sarà l'algoritmo di riferimento per le prestazioni nei due esperimenti con approccio tramite programmazione lineare intera

```
Put node_start in the OPEN list with f(node_start) = h(node_start) (initialization)
while the OPEN list is not empty {
  Take from the open list the node node_current with the lowest
       f(node_current) = g(node_current) + h(node_current)
  if node_current is node_goal we have found the solution; break
  Generate each state node_successor that come after node_current
  for each node_successor of node_current {
    Set successor_current_cost = g(node_current) + w(node_current, node_successor)
    if node_successor is in the OPEN list {
      if g(node_successor) ≤ successor_current_cost continue (to line 20)
    } else if node_successor is in the CLOSED list {
      if g(node_successor) ≤ successor_current_cost continue (to line 20)
      Move node_successor from the CLOSED list to the OPEN list
    } else {
      Add node_successor to the OPEN list
      Set h(node_successor) to be the heuristic distance to node_goal
   }
    Set g(node_successor) = successor_current_cost
    Set the parent of node_successor to node_current
  }
  Add node_current to the CLOSED list
}
if(node_current != node_goal) exit with error (the OPEN list is empty)
```

Figura 1.1: Pseudocodice A* [9]

Dove $OPEN$ consiste in nodi che sono stati visitati ma non espansi e $CLOSED$ consiste in nodi che sono stati visitati ed espansi

1.3 Programmazione lineare intera

La programmazione lineare intera[8] (PLI) si occupa del problema della minimizzazione o massimizzazione di una funzione lineare di più variabili in cui una o più variabili possono assumere soltanto valori interi, la funzione è soggetta a vincoli di disuguaglianza e uguaglianza lineari

$$\begin{cases} \min \ \mathrm{c}^T x \\ Ax \geq \mathrm{b} \\ x \in Z^n \end{cases}$$

è possibile anche un'altra formulazione in cui solo alcune variabili sono vincolate ad assumere un valore intero, nota come programmazione lineare mista (MILP)

$$\begin{cases} \min \ \mathrm{c}^T x \\ Ax \geq \mathrm{b} \\ \mathrm{x}_1 \in Z^{n_1}, x_2 \in R^{n_2} \end{cases}$$

$$con\ x = (x_1 x_2)\ e\ n = n_1 + n_2$$

$min\ c^T x$ rappresenta la funzione obiettivo, Ax≥b i vincoli e x le variabili del problema, più precisamente A è una matrice (m × n), b un vettore colonna m-dimensionale, c un vettore riga n-dimensionale e x un vettore colonna n-dimensionale. Andando ad osservare i problemi PLI e MILP da un punto di vista geometrico, facendo corrispondere l'insieme dei vincoli del problema a degli iperpiani che tagliano lo spazio degli stati delle variabili utilizzate, si riscontra che le soluzione ottime si trovano in corrispondenza dei vertici del politopo che si forma. L'algoritmo del simplesso va proprio a passare tutti i vertici del politopo valutando quale di essi abbia valore migliore rispetto alla funzione obiettivo.

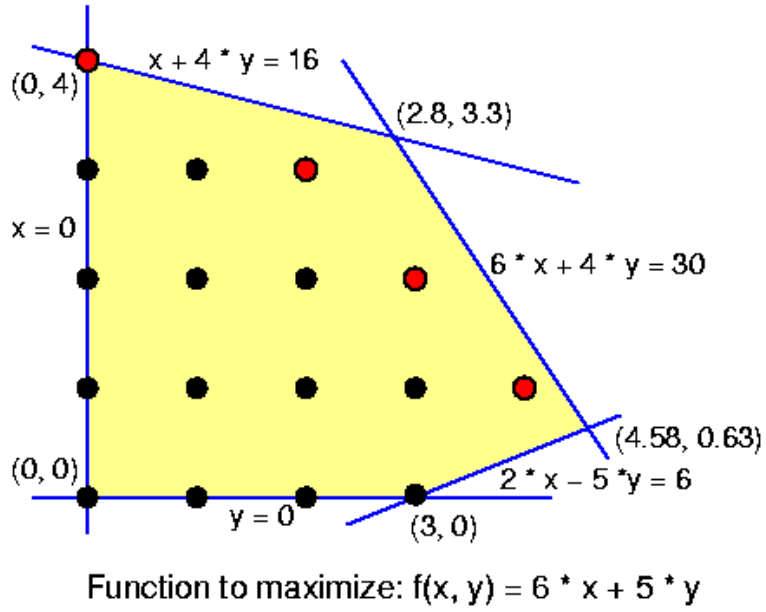

Figura 1.2: Esempio politopo [19]

I problemi di programmazione lineare PLI e MILP sono adatti alle applicazioni caratterizzate dalla necessità di scegliere tra un numero finito di alternative, modellando problemi complessi con buoni risultati e per questo è possibile pensare ad un loro utilizzo per la risoluzione di problemi di AI planning. Nell'ipotesi in cui la regione ammissibile di PLI o MILP è composta da un insieme finito di punti, teoricamente potrebbe essere sempre possibile risolvere il problema calcolando il valore della funzione obiettivo in ogni punto ammissibile per poi scegliere quello che la minimizza.

1.3.1 Branch and Bound

Una strategia usata dal risolutore è il Branch and Bound, che opera su un rilassamento lineare del problema senza tener conto del vincolo di interezza. La strategia è basata sulla suddivisone del problema in un certo numero di sottoproblemi la cui totalità rappresenta il problema iniziale. Se fin da subito la soluzione trovata dovesse essere intera si può concludere avendo trovato una soluzione ottima, altrimenti si procede operando sui sottoproblemi introducendo un vincolo aggiuntivo in modo da restringere lo spazio di ricerca, ripetendo le stesse operazioni fatte inizialmente; si itera così il processo fino a trovare una soluzione ottima o impossibile. Il procedimento viene solitamente rappresentato mediante un albero decisionale. Per l'efficienza di tale metodo è importante che una buona parte dei sottoproblemi possa essere risolta senza ricorrere ad ulteriori

suddivisioni, per questo viene calcolato un upper bound sul valore della soluzione del sottoproblema stesso in modo da evitare espansioni inutili [15]

Per implementare un algoritmo BB per uno specifico problema, bisogna determinare i seguenti elementi essenziali[3]:

1. Regole di Branching: come costruire l'albero delle soluzioni.
2. Calcolo del Bound: come valutare i nodi.
3. Regola di Fathoming: come chiudere e definire visitati i nodi.
4. Regole di esplorazione: definire le priorità di visita dei nodi aperti.
5. Come valutare una o più soluzioni ammissibili.
6. Criteri di stop: condizioni di terminazione dell'algoritmo.

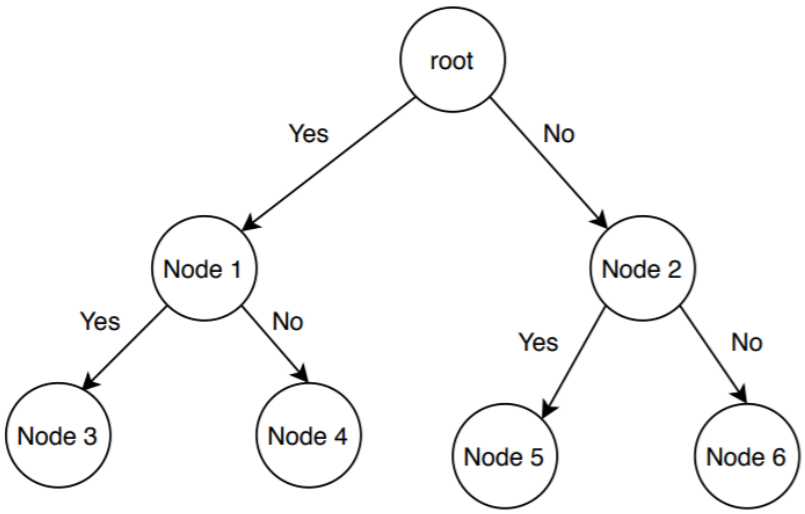

Figura 1.3: Esempio Branch and Bound [17]

1.3.2 Branch and Cut

Una strategia usata dai risolutori quando la soluzione del vertice del politopo è non intera è il Branch and Cut. Questa strategia combina il Branch and Bound e il metodo dei piani di taglio, il principio alla base consiste nell'aggiungere nuovi vincoli per ottenere soluzioni intere, aggiungendo iperpiani di taglio al politopo con lo scopo di ridurre lo scarto tra i vertici, ottenendo soluzioni intere; i piani di taglio vengono aggiunti in modo tale da preservare la soluzione intera ottimale

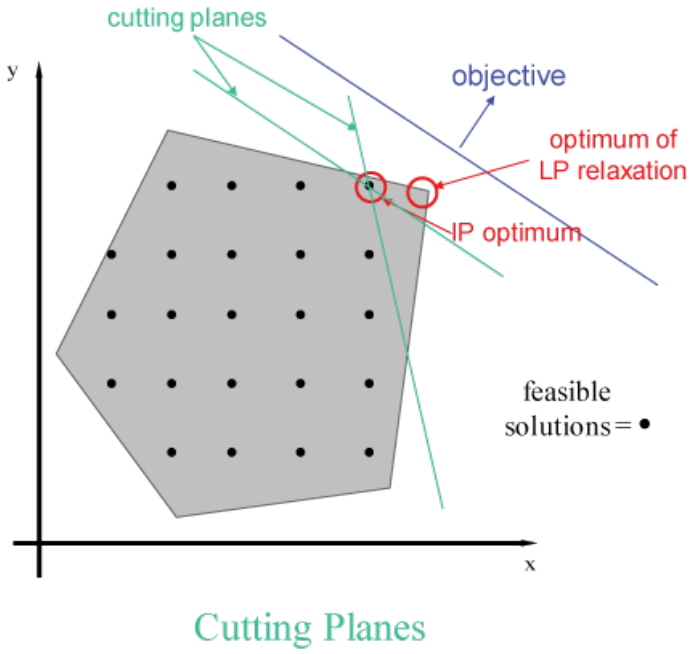

Figura 1.4: Esempio Branch and Cut [5]

1.4 Prova di ottimalità

Per definire ottima e valida una soluzione è necessario provare che la soluzione in questione sia la migliore possibile. I risolutori tengono in memoria sia della migliore soluzione intera definita come "incumbent" che del miglior rilassamento dei nodi alla frontiera. Si utilizzano nella definizione di:

1. Gap primale: differenza incumbent attuale e valore ottimo ricercato
2. Gap duale: differenza valore ottimo e miglior rilassamento

Per definire la soluzione come la migliore, incumbent e rilassamento devono convergere e in tal caso il valore dell'incumbent rappresenta la miglior soluzione possibile.

1.5 Single State Change Formulation

Il modello utilizzato nel corso degli esperimenti è il *Single State Change Formulation*[10]. Questa formulazione va ad utilizzare il parallelismo di Graphplan [11]. Graphplan è un pianificatore per problemi di tipo STRIPS, costruito attraverso una struttura dati detta planning graph che viene espansa ad ogni passo dell'algoritmo. Tra le sue caratteristiche introduce: il fattore tempo nel costruire un piano esplicitamente, produzione del

piano più corto possibile, proprietà tipiche dei risolutori lineari e non lineari, rispettivamente esegue early commitment e produce piani parzialmente ordinati. Il planning graph è un grafo diretto a livelli in cui:

1. I nodi sono raggruppati in livelli
2. Gli archi connettono i nodi ai livelli adiacenti

Nello specifico della modellazione da ora abbreviata con 1SC, per ogni fase di piano è consentita al massimo un cambio di stato per ciascuna variabile di stato. Le transizioni di stato di ciascuna variabile di stato sono rappresentate da percorsi in una rete ben definita. I nodi appaiono suddivisi in livelli e corrispondo ai possibili valori che le possibili variabili di stato possono assumere nei diversi step temporali. I livelli totali saranno $T+1$ dove T rappresenta la lunghezza massima del piano, il +1 è dovuto allo stadio iniziale. Gli archi corrispondo ai possibili cambiamenti di stato o persistenza di un valore. Nel corso del tempo le variabili passano da un valore al successivo fino al raggiungimento del livello finale che non necessariamente sarà nel livello T. Il percorso migliore nella rete rappresenterà proprio la soluzione cercata.

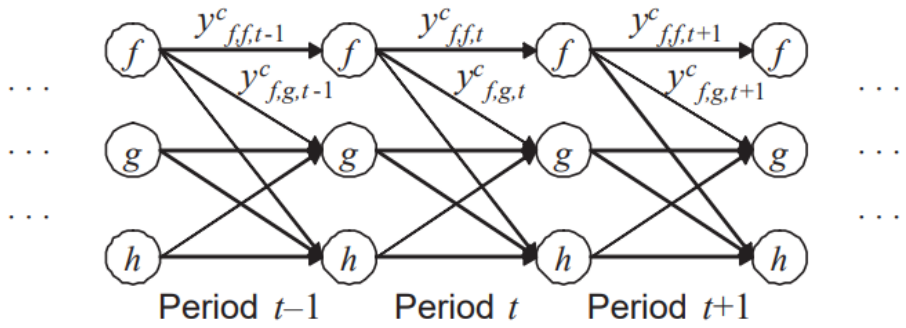

Figura 1.5: Esempio rete [10]

Il modello 1SC è il seguente:

$$min \sum_{t=1}^{T} \sum_{a \in A} cost_a \, z_{a,t} \tag{1.1}$$

$$\sum_{g \in D_v} y^v_{f,g,1} = \begin{cases} 1 & se \; f \in I_v \\ 0 & altrimenti \end{cases} \qquad \forall v \in V, g \in D_v \tag{1.2}$$

$$\sum_{h \in D_v} y^v_{g,h,t+1} = \sum_{f \in D_v} y^c_{f,g,t} \qquad \forall v \in V, \forall g \in D_v, 1 \leq t \leq T-1 \tag{1.3}$$

$$\sum_{f \in D_v} y^c_{f,g,T} = 1 \qquad \forall v \in G, g \in G_v \tag{1.4}$$

$$\sum_{a \in A : (f,g) \in SC_a(v)} z_{a,t} = y^v_{f,g,t} \qquad \forall v \in V, \; f, g \in D_v, f \neq g, 1 \leq t \leq T \tag{1.5}$$

$$z_{a,t} \leq y^c_{f,f,t} \qquad \forall v \in V, f \in D_v, a \in A, (f,f) \in PR_a(v), 1 \leq t \leq T \tag{1.6}$$

Prima di chiarire nel dettaglio il ruolo dei vincoli e della funzione obiettivo è meglio definire alcuni elementi utilizzati all'interno della formulazione:

- I rappresenta lo stato iniziale del problema, in cui ogni ad ogni variabile è assegnato un valore
- G rappresenta lo stato finale del sistema, non è detto che ad ogni variabile sia assegnato un valore finale

- $T \in \mathbb{N}$ rappresenta il numero massimo di step della risoluzione
- $V = \{v_1, v_2, ..., v_n\}$ rappresenta l'insieme delle variabili di stato, aventi ognuna un dominio D_v finito.
- Sono possibili due tipi di assegnamento:
    1. Assegnamento completo $s(v)$, in cui ad ogni variabile è assegnata un determinato valore appartenente al suo dominio, un esempio è lo stato I
    2. Assegnamento parziale, in cui alcune variabili non hanno un valore specificato, un esempio è lo stato G (anche se in realtà G può anche assumere uno stato con tutti assegnamenti completi)

- $A = \{a_1, a_2, ..., a_n\}$ rappresenta l'insieme delle azioni possibili. Ogni azione ha una precondizione per essere eseguita, un effetto e un costo in seguito indicati con: $pre(a), eff(a), cost(a)$
- $z_{a,t} \in \{0, 1\}$, $a \in A$ e $1 \leq t \leq T$, z assume valore 1 se l'azione a viene eseguita al tempo t, 0 altrimenti.
- $y^v_{f,g,t}$, $v \in V$, $f, g \in D_v$, $1 \leq t \leq T$, y assume valore 1 se la variabile di stato v passa dal valore f al valore g al tempo t, 0 altrimenti.

FUNZIONE OBIETTIVO 1.1

L'equazione descrive l'obiettivo del problema, i vincoli già da soli garantiscono la fattibilità del problema ma l'uso di una funzione obiettivo influenza le prestazioni poichè determina una direzione di ricerca. Si vuole ridurre al minimo il numero di azioni da eseguire

STATO INIZIALE 1.2

Il vincolo fa si che la somma in un determinato nodo al tempo iniziale degli archi uscenti sia 1 se è un valore definito nello stato iniziale I, 0 altrimenti

PROPAGAZIONE DELLO STATO 1.3

Il vincolo fa si che la somma degli archi entranti in un determinato nodo al tempo t sia uguale alla somma degli archi uscenti al tempo $t + 1$, in modo che i cambiamenti di stato delle variabili si propaghino in ogni step temporale

STATO FINALE 1.4

Il vincolo fa si che la somma degli archi entranti in un nodo al tempo finale T sia uguale a 1, per tutte le variabili che hanno uno stato finale non parziale

STATE CHANGE 1.5
Per ogni azione $a \in A$ viene definito lo state changes SC_a che rappresenta un cambio di valore su una variabile in seguito ad una azione, rappresentato da una tupla ($v \in V, f \in pre(a), g \in eff(a)$). Si va a formare un vincolo che collega tutti gli effetti del cambiamento di stato di un'azione sui corrispondenti archi. La sommatoria a sinistra dell'espressione ha lo scopo di impedire che più azioni interferiscano tra loro, solo una azione può provocare il cambiamento di stato nel determinato step temporale. È il primo vincolo che mette in relazione le variabili z e y, ovvero tra le variabili di azione e quelle di stato.

STATE PREVAIL 1.6
Per ogni azione $a \in A$ viene definito lo state prevail PR_a che rappresenta il mantenimento costante di un valore di una variabile nel momento in cui viene eseguita un'azione, rappresentato da una tupla ($v \in V, f \in pre(a)$). Questo vincolo afferma che se l'azione a è eseguita al tempo t ($z_{a,t} = 1$), allora prevale il valore f all'interno della variabile di stato v durante lo step t. Il vincolo impone come limite inferiore alle variabili di stato il valore della variabile d'azione.

1.5.1 Multiply Actions

Il modello 1SC viene affrontato con una particolare strategia detta $Multiply\ Action$ in seguito abbreviata con MA. Comprende una strategia volta ad evitare il problema di un eventuale valore indefinito, ossia quando $(v \in pre_a) = \emptyset$ e $(v \in eff_a) = valore$, andando a creare delle nuove azioni a partire da quella anomala che origina il problema. A causa del vuoto nelle pre_a MA creerà un certo numero di variabili aventi tutti i possibili valori in corrispondenza del relativo dominio, da inserire all'interno delle pre_a mentre il valore in seguito all'azione di eff_a resta invariato. Il risolutore così facendo andrà a scegliere ed utilizzare la variabili con la pre_a che più si addice alla specifica situazione Questa strategia tuttavia porta ad un possibile notevole aumento delle variabili di azione in corrispondenza di una elevata dimensione del dominio ma non porta all'aumento dell'orizzonte temporale

2

Esperimento

Nel capitolo sono esposti gli esperimenti e le risorse utilizzate per la risoluzione e la creazione del modello

2.1 Istanze

Negli esperimenti sono state utilizzate 150 istanze di problemi, tutte appartenenti alla classe MICONIC. Nel primo esperimento in particolare, sono state utilizzate 141 istanze poichè alcune erano mancanti all'interno delle risorse di tipo .sol, si rimanda al capitolo 3 per approfondimento. Le varie istanze sono numerate e ordinate secondo una specifica denominazione: $miconic - si - j$, con $i \in \{1, 2, ..., 30\}$ e $j \in \{0, 1, ..., 4\}$. Ognuno dei 150 problemi è definito da variabili con dominio $D_v \in \{0, ..., N\}$. In corrispondenza dell'aumentare dei due indici il problema aumenta di dimensione. Il parser utilizzato è $Fast\ downward$[4] che permette di oviare in parte alla grammatica scritta in PDDL dei problemi di IA planning.

2.2 Risorse

Tutti gli esperimenti sono stati eseguiti su un computer avete un processore Ryzen5-3500U, 8Gb di ram e sistema operativo Ubuntu 22.04.1. Sono state utilizzate varie funzioni della libreria standard di Python[14], pandas[13], matplotlib[18]. Infine Visual Studio Code[12] come editor per la creazione del codice

2.2.1 CPLEX E PYOMO

L'esperimento è stato sviluppato utilizzando il linguaggio di programmazione Python nella versione 3.10.6. Per la creazione del modello si è utilizzata la libreria Pyomo nella versione 6.4.2.[2]. Tale libreria open-source è specializzata nella formulazione di modelli per problemi di ottimizzazione che comprendono quelli di programmazione lineare intera. Pyomo viene utilizzato esclusivamente per la definizione e creazione del modello da passare al risolutore e non ha nulla a che vedere con la risoluzione del problema. L'utilizzo di Pyomo permette di sfruttare la potenza espressiva e moderatamente semplice del linguaggio Python con risolutori efficienti ma di difficile utilizzo. Il risolutore utilizzato negli esperimenti è CPLEX nella versione 22.1.0.0 [6], sviluppato da IBM. CPLEX Optimizer fornisce risolutori della programmazione matematica flessibili e con prestazioni elevate per la programmazione lineare, a interi misti, quadratica e per i problemi di programmazione con vincoli quadratici.

2.3 TEMPI

La formulazione 1SC richiede un tempo massimo T per la risoluzione delle varie istanze. Questo dato è stato ottenuto dal planner di riferimento in quanto non avrebbe senso andare a superare tale valore con le strategie utilizzate. Tuttavia nel corso degli esperimenti per quanto riguarda la risoluzione, è stato impostato al risolutore un timelimit di 10 minuti per ogni istanza. Se si dovesse superare tale limite temporale si assegna un tempo di risoluzione di 600 secondi indipendentemente dal tempo ancora necessario per il calcolo della soluzione. Per ogni istanza è stato calcolato il tempo di risoluzione, da quando il risolutore inizi ad operare sul problema fino alla sua risoluzione.

2.4 DESCRIZIONE ESPERIMENTO

Entrambi gli esperimenti rientrano nel $proof\ of\ concept$, ovvero un progetto con lo scopo di dimostrare la fondatezza o fattibilità di concetti e principi alla base dell'esperimento. Non c'è l'obiettivo di competere con soluzioni già esistenti, anche commerciali, ma piuttosto dimostrare la validità dell'esperimento per eventualmente proseguire in seguito con maggior attenzione nel caso si ottenessero risultati positivi.

2.4.1 Esperimento A: formulazione MA senza soluzione in input

Nel primo esperimento sono state eseguite tutte le istanze utilizzando il modello 1SC con strategia MA, tenendo traccia delle azioni eseguite e del tempo di risoluzione considerando sempre un timelimit di 10 minuti.

2.4.2 Esperimento B: formulazione MA con soluzione in input

Nel secondo esperimento sono state inizialmente calcolate le soluzioni ottime dal planner. Delle 150 istanze MICONIC ne sono state risolte 141, ovvero il 94% usate quindi come soluzione ottima. In particolare le istanze che non sono state risolte dal planner sono le: 19-2, 20-0, 23-1, 23-3, 25-2, 26-1, 27-0, 28-1, 30-3 aventi l'indicizzazione indicata al punto 2.1. In seguito si è ripetuto il processo descritto nella prima parte ma con una differenza sostanziale nella risoluzione. Prima del processo di risoluzione, è stato passata in input la soluzione ottima calcolata precedentemente dal planner di riferimento, in modo da suggerire al risolutore una soluzione, segnalando quali azioni compiere in ogni step temporale impostando le variabili z corrispondenti a 1. Così facendo i tempi, soprattutto nelle istanze con indici maggiori, dovrebbero calare notevolmente

3

Risultati sperimentali

Nel capitolo vengono presentati i risultati degli esperimenti effettuati.

3.1 Bias temporale

Nel primo esperimento è stato utilizzato un timelimit pari a 600 secondi. La scelta del limite temporale è in parte legata alla macchina utilizzata nel corso degli esperimenti, è possibile ridurlo se dotati di una macchina più performante. Tuttavia raramente il timelimit è stato raggiunto nel modello con formulazione MA e soluzione ottima passata al risolutore, ricavata dal planner di riferimento. Solo poche istanze sono state bloccate al raggiungimento dei 10 minuti. La maggior parte come si può osservare dal grafico 3.4 sono state inferiori al minuto. Risultati ben diversi dal modello senza soluzione ottima. Andando a considerare il tempo trascorso dall'esecuzione del programma fino al suo completamento si riscontra che la maggior parte del tempo, sopratutto in corrispondenza delle istanze con indice maggiore, è stato impiegato per la costruzione del modello e non tanto per la sua risoluzione, fenomeno anch'esso riducibile con una macchina più performante

3.2 Operazioni sui dati

Una volta ottenuti i dati, sono state effettuate delle operazioni per poter eseguire una analisi più dettagliata e precisa.

3.2.1 Media geometrica con shift

Si è scelto di utilizzare la media geometrica shiftata rispetto a quella aritmetica poichè è meno sensibile agli outlier, elementi sicuramente presenti, andando a considerare tempi che spaziano da millesimi di secondo a minuti. Al contrario tale media è particolarmente sensibile a numeri piccoli (in questo caso rappresentati da i millesimi di secondo), senza contare che il risolutore e la macchina compiono sicuramente degli errori e approssimazioni con tali numeri. La soluzione quindi è usare uno shift pari ad un certo delta, eseguire i calcoli per poi sottrarre lo shift al risultato ottenuto.

- Tempi: lo shift s è stato scelto pari a 1 secondo

$$\sqrt[150]{\prod_{i=1}^{150}(t_i + s)} - s$$

- Nodi lo shift N è stato scelto pari a 10 nodi

$$\sqrt[150]{\prod_{i=1}^{150}(n_i + N)} - N$$

3.3 Note sulla formulazione

In entrambe le versioni non sono state utilizzate e risolte le istanze MICONIC a partire dalla 26.0, dato che i tempi di costruzione del modello stesso erano eccessivamente lunghi. Tuttavia arrivando al problema 25.4 si ha una quantità sufficiente di elementi per poter valutare i risultati in modo esaustivo.

3.3.1 Formulazione senza soluzione in input

La percentuale di istanze risolte nella versione senza soluzione passata in input è: 74‰
A partire dalla soluzione 18.1 il risolutore non è stato in grado di trovare una soluzione, tuttavia non è detto che questa non esista, a causa della presenza del timelimit impostato a 600 secondi. All'interno di questo intervallo temporale non si è andati oltre il node radice, non disponendo dunque di nessuna soluzione; in alcuni casi (ES 17.3) rimuovendo il limite una soluzione è stata trovata ma in tempi decisamente troppo lunghi che vanno

totalmente a vanificare l'approccio utilizzato. Dunque nella percentuale di istanze non risolte si vanno a considerare sia l'impossibilità nel ricavare una soluzione che il limite di tempo imposto che può in alcuni casi bloccare l'esplorazione. L'errore ritornata dal risolutore "ValueError: Cannot load a SolverResults object with bad status: error" va a indicare che non è possibile accedere ad una soluzione poichè il problema è non ammissibile, illimitato oppure non esiste una soluzione.
Le istanze 18.2,3,4 19.0,4 20.2,3 tuttavia sono state risolte pur avendo indici superiori a quello indicato sopra.

3.3.2 FORMULAZIONE CON SOLUZIONE IN INPUT

La percentuale di istanze risolte nella versione con soluzione passata in input è: 94% Il risolutore in questo caso è sempre riuscito a trovare una soluzione, il timelimit di 600 secondi è stato raggiunto solo in poche istanze (ES 17.3). Poichè sempre in possesso di almeno una soluzione per come è stato costruito il modello, anche una volta raggiunto tale limite temporale, non è mai stato segnalato errore o soluzione non trovata. Nel caso in cui infatti non fosse stata trovata un'altra soluzione ottimale, il risolutore avrebbe riportato la soluzione passata in input dal planner di riferimento.

3.4 RISULTATI NUMERICI

Percentuali di risoluzione

Le percentuali di risoluzione si dividono in due parti: una considera tutte le 150 istanze MICONIC, la seconda le prime 125 ovvero fino alla 25.4 Il calcolo è stato fatto utilizzando la seguente formula:

$$\frac{istanze\ risolte}{istanze\ totali}$$

Totale 150

- Planner A^*: 94%
- Formulazione MA senza soluzione in input: 74%
- Formulazione MA con soluzione in input: 94%

Totale 125

- Planner A^*: 96%

- Formulazione MA senza soluzione in input: 70.4%
- Formulazione MA con soluzione in input: 96%

Media Geometrica shiftata

In accordo con la formula indicata nel punto 3.2.1, i risultati ottenuti sono:

Formulazione senza soluzione in input

- Tempi: 22.26 secondi
- Nodi: 46.3

Formulazione con soluzione in input

- Tempi: 8.79 secondi
- Nodi: 5.55

3.5 Grafici

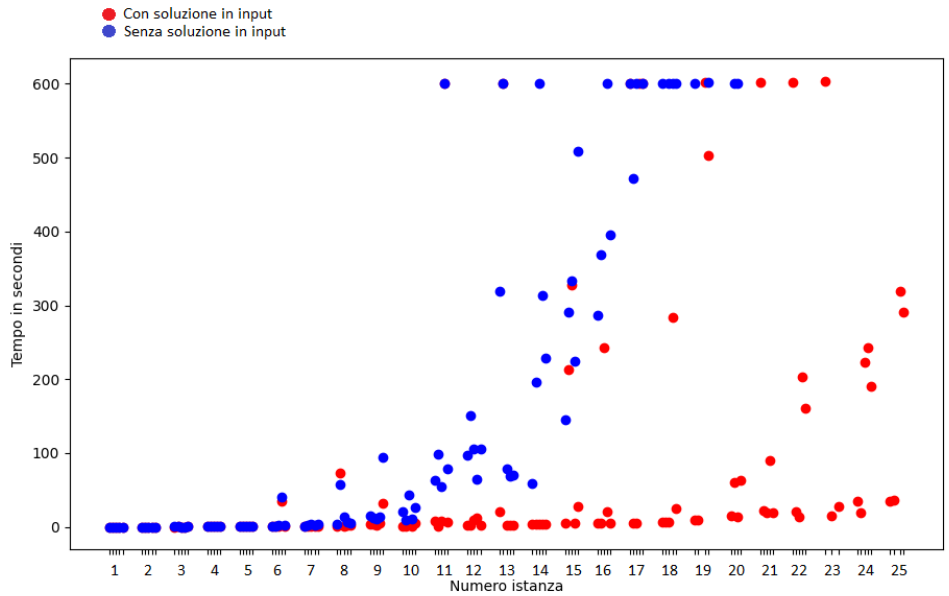

Figura 3.1: Confronto tempi formulazione con soluzione in input e senza input

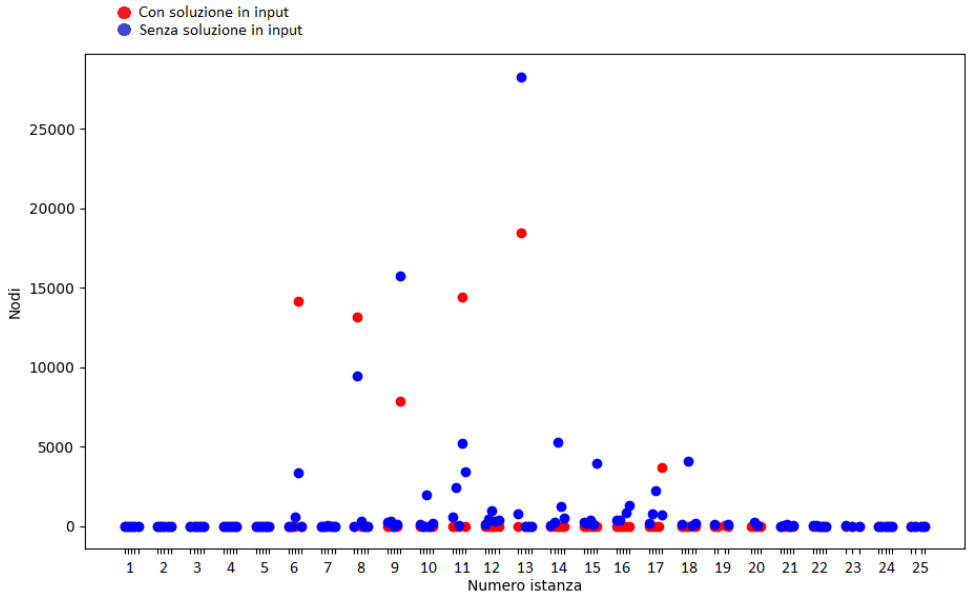

Figura 3.2: Confronto nodi formulazione con soluzione in input e senza input

Seguono i singoli grafici relativi ai tempi e nodi senza soluzione passata in input e con soluzione passata in input

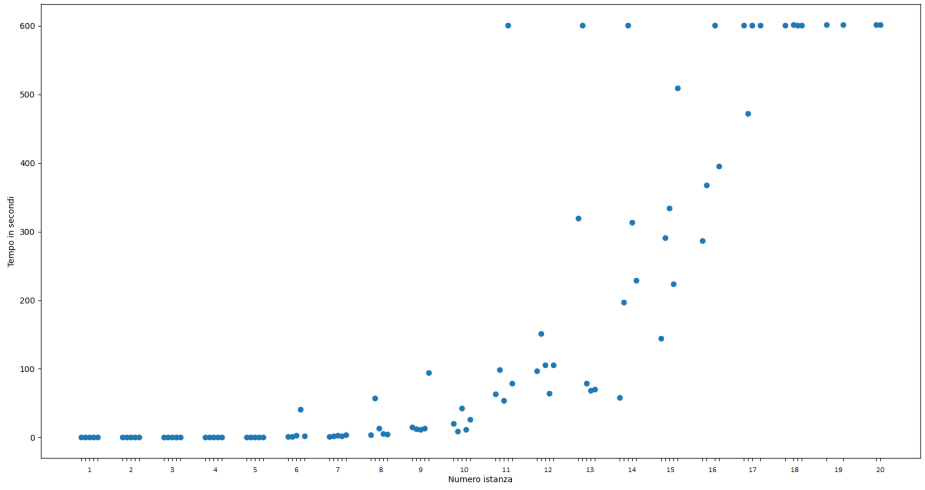

Figura 3.3: Numero istanza e tempo risoluzione - senza soluzione in input

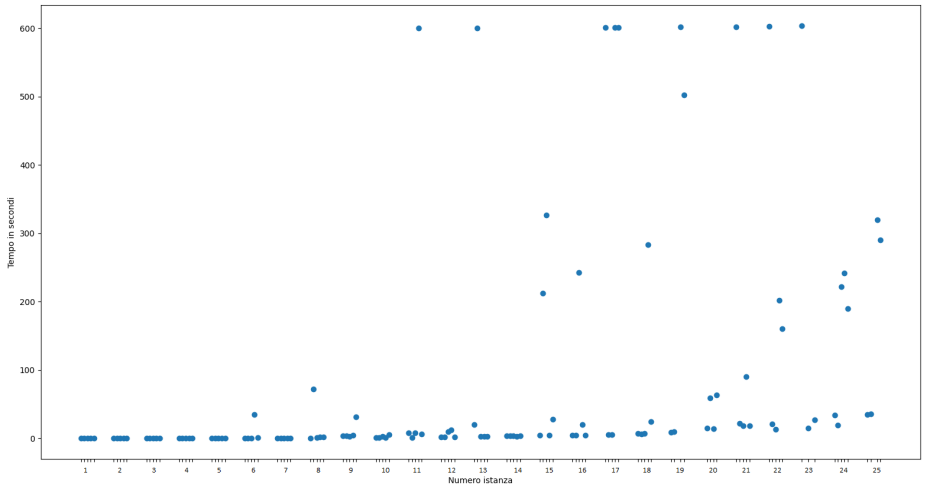

Figura 3.4: Numero istanza e tempo risoluzione - con soluzione in input

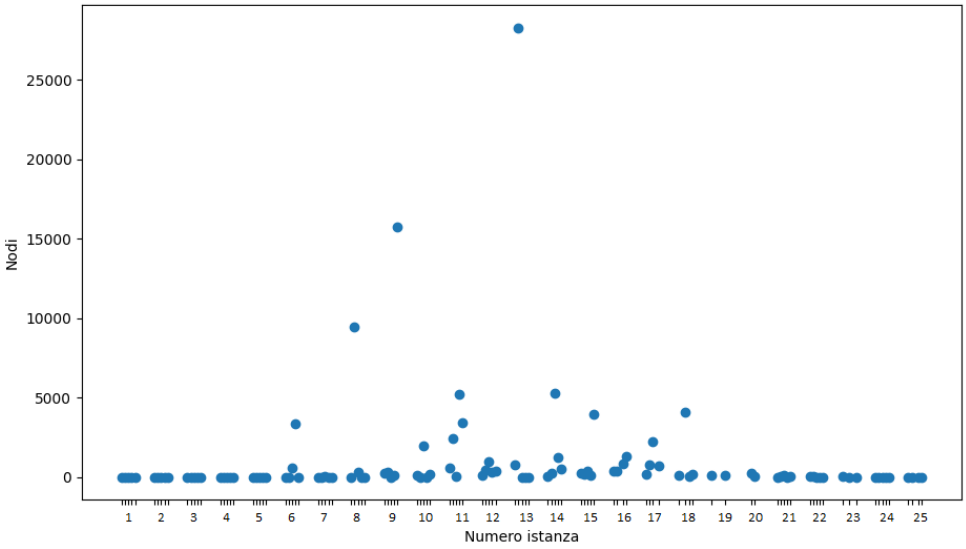

Figura 3.5: Numero istanza e numero nodi - senza soluzione in input

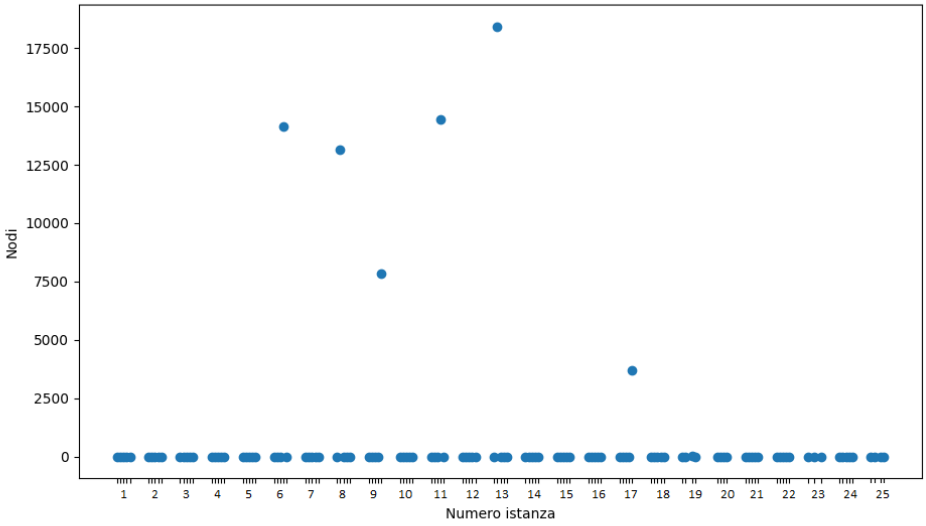

Figura 3.6: Numero istanza e numero nodi - con soluzione in input

4

Analisi e conclusioni

Nel capitolo verranno analizzati i dati ottenuti, riportati graficamente nel capitolo 3

4.1 Analisi degli esperimenti e performance

Entrambe le formulazioni non hanno risolto le istanze a partire dalla 26.0 compresa, il problema è dovuto ai tempi troppo lunghi della costruzione del modello e non alla sua risoluzione. L'analisi dei dati ottenuti dunque è relativo alle prime 125 istanze. Con l'uso di una macchina aventi prestazioni migliori il problema relativo alla costruzione è facilmente risolvibile.

4.1.1 Analisi formulazione MA con soluzione in input

La formulazione MA con la soluzione passata in input è riuscita a risolvere tutte le istanze, ad eccezione di 9, ovvero quelle non risolte da A*; non disponendo dunque di una soluzione ammissibile da passare al risolutore. Il timelimit pari a 600 secondi è stato raggiunto da poche istanze, la maggior parte hanno risolto il problema in tempi molto brevi come si può vedere dal grafico 3.4. Anche una volta raggiunti i 10 minuti il risolutore disponeva sempre di una soluzione, ovvero quella passata in input, di conseguenza se nell'arco di tempo disponibile non fosse riuscito a trovarne un'altra ciò non sarebbe risultato un problema. Dal grafico 3.6 si può osservare come la quasi totalità delle istanze sia stata risolta andando poco oltre il nodo radice, assumendo un andamento lineare e costante, ad eccezione di 6 istanze ovvero il 0.05%

4.1.2 Analisi formulazione MA senza soluzione in input

La formulazione MA senza la soluzione passata in input non è riuscita a risolvere tutte le istanze. In particolare dalla 20.4 il risolutore non è stato in grado di trovare almeno una soluzione nell'arco dei 10 minuti di tempo. Come si può vedere dal grafico 3.3 le istanze con indice minore hanno avuto tempi di risoluzione molto simili alla versione con soluzione in input, al contrario con l'aumentare delle dimensioni dei problemi i tempi sono aumentati, seguendo una certa regolarità; indici maggiori e tempi più lunghi. Le istanze con indice maggiore, a partire dalla 15.0 hanno in gran parte raggiunto il timelimit. Dal grafico 3.5 si può osservare un andamento verso il nodo radice ma meno marcato rispetto alla versione con soluzione, il numero di nodi è sicuramente maggiore in questo caso.

4.2 Possibile miglioramento

Dati i risultati ottenuti con la formulazione MA con soluzione passata in input è possibile pensare ad un miglioramento per provare ad aumentarne ulteriormente le prestazioni. L'idea è provare a ridurre l'orizzonte temporale T calcolato dal planner di riferimento e passato alla formulazione MIP. È fattibile potenzialmente eseguire più azioni in contemporanea cercando dunque di ridurre al minimo il valore di T, ovvero diminuire il numero di step necessari eseguendo più azioni possibili in parallelo.

4.3 Conclusioni

La formulazione MA con soluzione passata in input ha sicuramente avuto performance migliori rispetto alla versione senza soluzione, oltre ad essere sempre riuscita a riportare una soluzione i tempi sono stati decisamente migliori. Bisogna tenere conto tuttavia che per ottenere la soluzione iniziale da passare in input e il numero di step T si è dovuto utilizzare un altro approccio, più efficace e performante. I tempi dunque complessivi nell'uso di tale approccio non sono quindi competitivi ma evidenziano come un approccio misto riesca ad ottenere dei buoni risultati e quindi non è da escludere la possibilità e l'utilizzo di approcci tramite la programmazione lineare per la risoluzione di problemi di AI Planning.

Bibliografia

[1] Alessandro Saetti. *PDDL*. URL: https://lpg.unibs.it/ia/lucidi_11-12/PDDL.pdf.

[2] Michael L. Bynum et al. *Pyomo–optimization modeling in python*. Third. Vol. 67. Springer Science & Business Media, 2021.

[3] De Giovanni. *Branch and Bound*. URL: https://www.math.unipd.it/~luigi/courses/ricop0809/ro_13.m04.BeB.01.pdf.

[4] *Fast Downward*. URL: https://www.fast-downward.org/.

[5] Gurobi Optimization. *Cutting plane*. URL: https://www.gurobi.com/resources/mixed-integer-programming-mip-a-primer-on-the-basics/.

[6] IBM. *Cplex optimizer*. URL: https://www.ibm.com/it-it/analytics/cplex-optimizer.

[7] IBM. *IA Planning*. URL: https://researcher.watson.ibm.com/researcher/view_group.php?id=8432.

[8] Laura Palagi. *Programmazione lineare intera*. URL: http://users.diag.uniroma1.it/~palagi/didattica/sites/default/files/cap11.pdf.

[9] Lluis Alseda. *A^* algorithm*. URL: https://mat.uab.cat/~alseda/MasterOpt/AStar-Algorithm.pdf.

[10] Menkes van den Briel,Thomas Vossen,Subbarao Kambhampati. *Reviving Integer Programming Approaches for AI Planning: Branch-and-Cut Framework*. URL: https://www.aaai.org/Papers/ICAPS/2005/ICAPS05-032.pdf.

[11] Michele Lombardi. *GraphPlan*. URL: http://www.lia.deis.unibo.it/Courses/AI/applicationsAI2008-2009/Lucidi/es-Graphplan.pdf.

[12] Microsoft. *Visual Studio Code*. URL: https://code.visualstudio.com/.

[13] NumFOCUS. *Pandas*. URL: https://pandas.pydata.org/.

[14] Python software fondation. *Python*. URL: https://www.python.org/.

[15] Silvano Martello. *Branch and Bound*. URL: http://www.or.deis.unibo.it/staff_pages/martello/Ricerca%20Operativa%20LS/BB.pdf.

[16] Peter Norvig Stuart Russell. *Artificial Intelligence: A Modern Approach*. 11 agosto 2021. mylab, 2015.

[17] Subham Datta. *Branch and Bound Algorithm*. URL: https://www.baeldung.com/cs/branch-and-bound.

[18] The Matplotlib development team. *MatplotLib*. URL: https://matplotlib.org/.

[19] Vincent Conitzer. *Politopo*. URL: https://courses.cs.duke.edu/fall12/compsci590.1/.