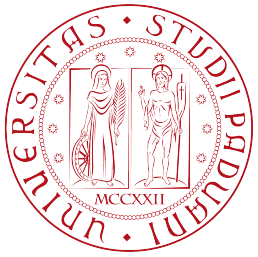


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in
Ingegneria Informatica


# Approcci a problemi di AI Planning tramite programmazione lineare intera


*Relatore:*
Prof. Domenico Salvagnin

*Laureando:*
Pietro Girotto
1216355



Anno Accademico 2021/2022

**Abstract**

I problemi di AI Planning sono pattern facilmente riconoscibili dal potere espressivo considerevole. Non vi è quindi sorpresa essi siano stati attentamente studiati per poter creare tool specifici e prestanti, adatti alla loro risoluzione.
Allo stato dell'arte si utilizzano strategie di ricerca euristica in quanto più efficienti. Tuttavia approcci di programmazione lineare intera sono stati usati con successo in ricerche precedenti, rendendo quindi importante esplorare tali metodi per la risoluzione di problemi di AI Planning.
In questo studio preliminare vengono quindi comparate due formulazioni MIP con l'algoritmo di ricerca $A^*$ implementato dal risolutore FastDownward. Risulta chiara la superiorità degli strumenti di ricerca euristici su quelli MIP, ciò nonostante vi sono delle riflessioni su possibili miglioramenti o analisi future che possano concludere l'eventuale efficacia di questi approcci.

# Indice

# Capitolo 1

# Introduzione

La risoluzione di problemi e l'automazione sono due dei più grandi promotori nel campo dell'informatica. Tanto più il problema si fa complesso, voluminoso o oscuro tanto più è ricercata la strategia di risoluzione. In questa tesi verranno esplorati diversi meccanismi per la risoluzione di problemi reali e dalla natura estremamente polivalente, valutando quali siano le strategie migliori di risoluzione e quali siano possibili orizzonti per il futuro.

## 1.1 Problemi di AI Planning

AI Planning, anche detto Automated Planning, è una branca dell'informatica devota allo studio e risoluzione di problemi dalle caratteristiche ben definite. Questi problemi devono avere uno stato iniziale, uno finale e un insieme di azioni che ci permettano di trasformare l'uno nell'altro [1] cercando la soluzione dal costo minore possibile.

Tale definizione, appunto perché estremamente generica, lascia spazio ad una moltitudine di applicazioni pressoché infinita, limitata esclusivamente dalla possibilità di tradurla in modelli matematici.
Nella loro forma più basica, i problemi di AI Planning sono definiti da un insieme di variabili $V = \{v_1, v_2, v_3, ...\}$ con cardinalità finita ed un insieme di azioni $A = \{a_1, a_2, ...\}$ anch'esso di cardinalità finita.
Ogni azione è poi definita a sua volta come una tupla $a_i := < pre(a_i), eff(a_i), cost_i >$ dove:

- $pre(a_i)$ sono le condizioni necessarie affinché $a_i$ sia applicabile
- $eff(a_i)$ sono gli effetti che l'azione $a_i$ genera se utilizzata

- $cost_i$ è il costo dell'azione $a_i$

Lo scopo è trovare una successione valida di azioni dal costo minimo per raggiungere uno stato finale detto *stato di Goal*.

## 1.2 Algoritmo $A^*$

L'algoritmo $A^*$ è estremamente utile nell'attraversamento di grafi. Questo è dovuto alle sue prestazioni temporali ottime [2] e la sua completezza.

$A^*$ è un algoritmo per la ricerca di path minimi all'interno di grafi pesati basato su una funzione euristica $g(n)$ ammissibile che stima la distanza dei diversi nodi dal goal finale.
Partendo da un nodo iniziale, ad ogni iterazione $A^*$ valuta ciascun nodo della frontiera[1] con la funzione $f(n)$ dove:

$$\begin{cases} f(n) = h(n) + g(n) \\ h(n) \rightarrow \textit{costo percorso fino al nodo ispezionato} \\ g(n) \rightarrow \textit{funzione euristica ammissibile} \end{cases} \tag{1.1}$$

L'algoritmo procede quindi a scegliere il nodo con $f(n)$ minore e ripete il passaggio precedente fino a trovare un percorso dal costo minimo che permetta l'attraversamento del grafo dal nodo iniziale al nodo goal.

Come si può intuire, $A^*$ si rivela particolarmente prono alla ricerca di soluzioni ottimali nei problemi di AI Planning. Viene infatti utilizzato in AI Planner moderni come opzione competitiva.
Nello specifico, $A^*$ sarà l'algoritmo di riferimento per i diversi indici di prestazione dei metodi di programmazione lineare intera che verranno misurati.

## 1.3 Mixed Integer Linear Programming

Un importante paradigma di programmazione è la programmazione lineare intera mista, più semplicemente detta MILP.

La programmazione lineare intera mista si occupa della risoluzione di problemi dalle seguenti caratteristiche:

- Deve presentare un numero di variabili e vincoli finito

[1] La frontiera viene intesa come l'insieme dei nodi aperti e non ancora ispezionati

- Le sue variabili devono essere esclusivamente reali o, al più, intere
- I vincoli devono essere esclusivamente lineari

Un problema MILP in forma standard viene descritto come:

$$\begin{cases} min\ \boldsymbol{c}^T\boldsymbol{x} \\ A\boldsymbol{x} = \boldsymbol{b} \\ \boldsymbol{x} \geq 0 \\ x_j \in \mathbb{Z} \quad j \in J \end{cases} \tag{1.2}$$

Dove $min\ \boldsymbol{c}^T\boldsymbol{x}$ viene chiamata *funzione obiettivo*, $A\boldsymbol{x} = \boldsymbol{b}$ e $\boldsymbol{x} \geq 0$ definiscono invece i vincoli lineari del problema, $x_j$ sono le eventuali variabili intere.
Nonostante possa sembrare un paradigma estremamente limitato e poco efficace, ha una capacità espressiva incredibilmente potente. È infatti possibile, tramite dovute accortezze, modellare problemi complessi su paradigmi di tipo MILP, permettendo quindi la risoluzione di tali problemi con strumenti specifici del caso.
Immaginando l'insieme di vincoli come iperpiani che tagliano lo spazio degli stati delle nostre variabili, ci si accorge che le soluzioni (intere o frazionarie che siano) del problema di ottimizzazione ricadono sempre nei vertici del politopo che si viene a definire.

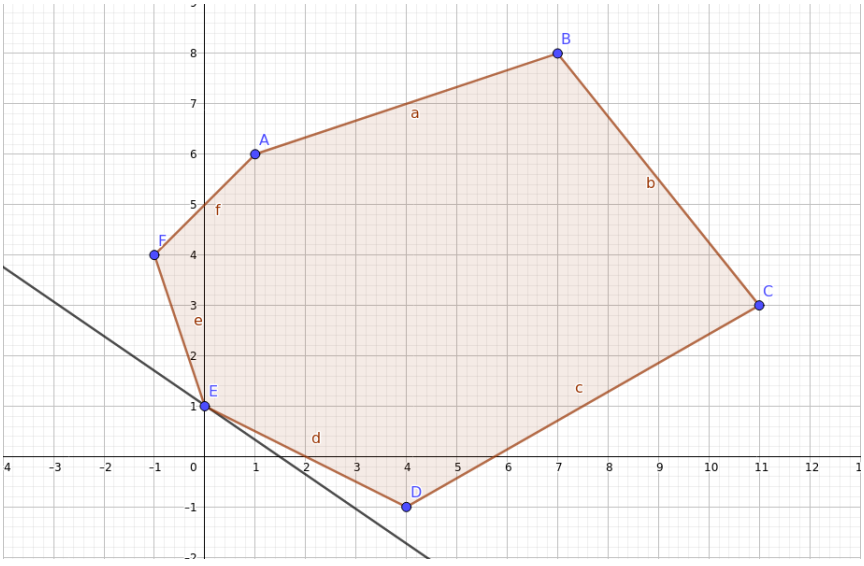


**Figura 1.1:** Il politopo (in arancio) e la funzione di ottimizzazione (in grigio)

L'algoritmo del simplesso [4] sfrutta questa peculiarità passando di vertice in vertice e valutando quale di essi abbia valore migliore per la funzione di ottimizzazione.

In caso si cercassero però soluzioni intere, non è generalmente dato che i vertici del politopo individuino una soluzione non frazionaria. In questo caso, una volta formulato il problema con vincoli lineari e variabili intere e/o reali, il risolutore scelto utilizzerà strategie di **Branch and Bound** e **Branch and Cut** per riuscire ad ottenere una soluzione ottima, o provare l'assenza di quest'ultima [3].

### 1.3.1 Branch and Bound

Una volta generato il problema, non sapendo come risolverlo per soluzioni intere si procede a creare un rilassamento lineare del problema iniziale che non presenti il vincolo di interezza. Poiché il problema attuale è risolvibile, si procede a individuarne la soluzione, anche qualora essa non dovesse risultare intera.
Se la soluzione dovesse essere fortunatamente intera, si ha trovato un valore ottimo e si può concludere. Più genericamente la soluzione non soddisferà i vincoli di interezza e sarà quindi necessario creare due sotto problemi con un vincolo aggiunto che restringa lo spazio di ricerca. Se, per esempio, dovessimo trovare come valore del rilassamento $x = 6.33$ procederemmo a creare due problemi (senza rilassamenti) con ciascuno un vincolo tra $x \geq 7$ oppure $x \leq 6$, permettendoci quindi di cercare ulteriori soluzioni.
Ottenuti questi sotto problemi non si fa altro che reiterare il processo fintanto che la soluzione trovata risulta ottima, infinita o impossibile.

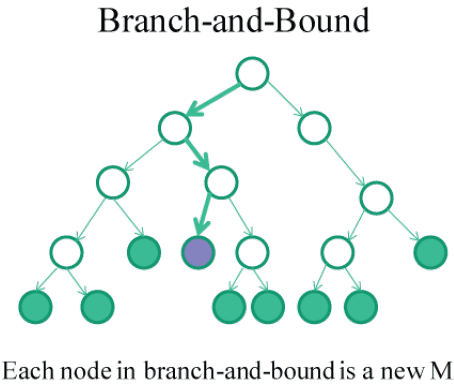


**Figura 1.2:** Esempio di albero B&B [3]

### 1.3.2 Branch and Cut

Una ulteriore strategia che i risolutori mettono in pratica è il Branch and Cut.
Branch and Cut è una strategia che mira ad aggiungere iperpiani di taglio al politopo originale con l'obiettivo di ridurre significativamente lo scarto fra i vertici

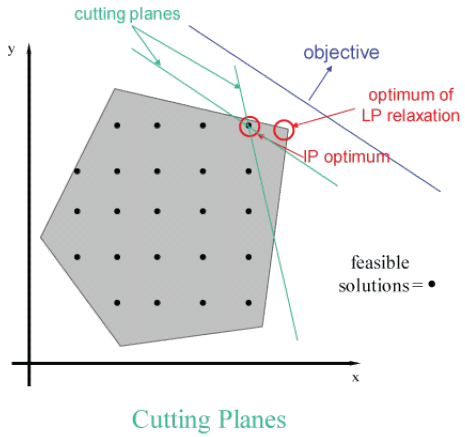


**Figura 1.3:** Esempio di B&C [3]

e le soluzioni intere del problema. In questa maniera risulta quindi possibile individuare subito una soluzione intera o perlomeno rendere più forti i rilassamenti, diminuendo quindi il numero di branch effettuati da B&B.

### 1.3.3 Prova di Ottimalità

Affinché la soluzione trovata possa essere considerata valida vi è necessità di provare che essa sia la migliore possibile.
Per fare questo, i diversi risolutori tengono traccia sia del miglior rilassamento tra i nodi della frontiera che della migliore soluzione intera trovata, quest'ultima anche detta **incumbent**. Fintanto che i due valori non convergono non sarà possibile stabilire se l'incumbent attuale sia la migliore soluzione possibile o solo una delle molteplici soluzioni valide.
Volendo rappresentare l'andamento nel tempo[2] di incumbent e miglior rilassamento sarà genericamente osservata una progressiva convergenza dei due valori al valore ottimale effettivo.

[2]di un problema di minimizzazione

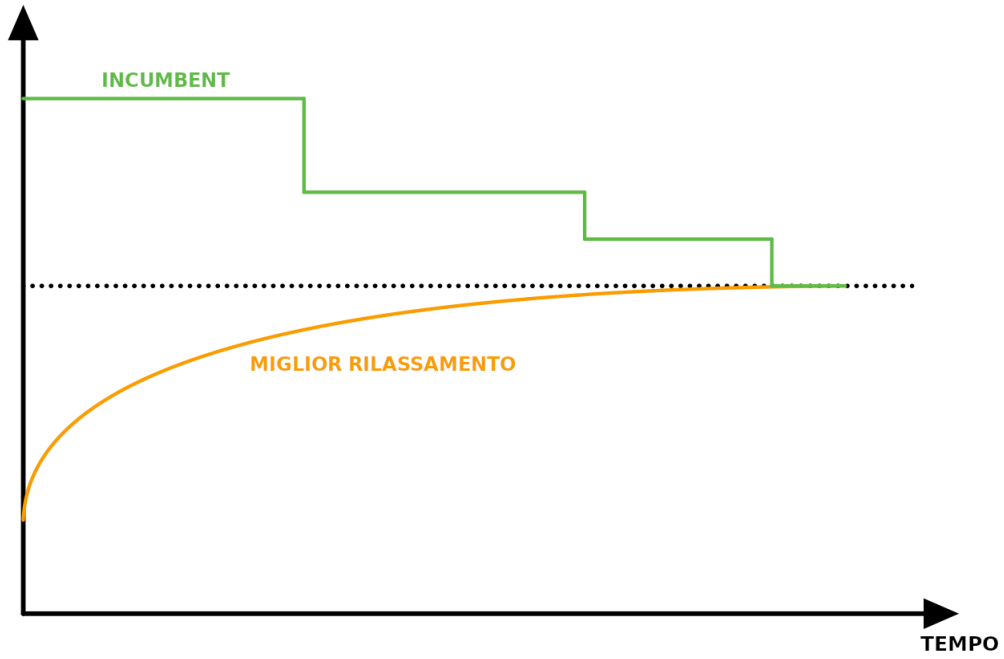


**Figura 1.4:** Esempio di un possibile andamento del rilassamento e dell'incumbent in un problema di minimo

Si introducono inoltre due definizioni:

- **GAP primale** definito come la differenza fra l'incumbent attuale e il valore ottimale ricercato
- **GAP duale** definito come la differenza fra il valore ottimale e il miglior rilassamento

## 1.4 Modellazione di problemi di AI Planning

Come precedentemente introdotto, sono necessari diversi accorgimenti affinché si possano modellare problemi reali in problemi di programmazione lineare intera. A seconda del modello matematico scelto e del problema affrontato, le strategie MILP possono soffrire di grandi fluttuazioni in performance. È importante quindi scegliere accuratamente il giusto modello per il giusto problema.
In questa tesi verrà affrontata una specifica modellazione di basici problemi di AI Planning chiamata *Single State Change Formulation* [5].
Prima di illustrare il modello stesso è necessario iniziare con alcune definizioni elementari:

- $C = \{c_1, ..., c_n\}$ è l'insieme delle *variabili di stato*, dove ogni variabile di stato ha un suo dominio $D_c$ di dimensione finita. Si definisce uno *stato* $s(c)$ come un assegnamento completo di $C$, ossia per ogni variabile in $C$ è data ad essa un valore del suo dominio $D_c$. Se l'assegnamento risulta parziale, ossia vi sono variabili il cui valore non è specificato, tale assegnamento si definisce *stato parziale*.

- $I$ è lo stato iniziale del sistema, priore a qualsiasi azione. Si noti che esso è per definizione un assegnamento completo.

- $G$ è lo stato parziale finale del sistema. In esso sono descritti i valori finali di una o più variabili di stato. Si noti che in quanto parziale non è necessario specificare un valore finale per ogni variabile.

- $A = \{a_1, ...a_m\}$ è l'insieme delle azioni (o operatori) $a_i$. Una azione viene definita come una tupla $a :=< pre(a), eff(a), cost(a) >$ i cui elementi rappresentano rispettivamente le precondizioni, gli effetti e il costo[3] dell'azione $a$. Per ogni azione $a \in A$ si definiscono le *state changes* di $a$ come $SC_a$ e le *prevail* come $PR_a$. $SC_a$ è un insieme di tuple $(c, f, g)$ tali che $s(c) = f \in pre_a \ and \ s(c) = g \in eff_a$ rappresentando quindi un cambio di valore da $f$ a $g$ sulla variabile $c$ effettuato dall'azione $a$. $PR_a$ rappresenta invece le tuple $(c, f)$ tali che $s(c) = f \in pre_a \ and \ \forall g \neq f \rightarrow s(c) = g \notin eff_a$, ossia una azione in cui una variabile mantiene costante il suo valore $f$ nel momento in cui viene effettuata l'operazione $a$. Non è possibile per alcuna azione $a \in A$ avere più di un effetto di state change o prevail su una certa variabile $c \in C$.

- $T \in \mathbb{N}$ che definisce il numero massimo di steps del piano di risoluzione.

- $x_{a,t} \in \{0, 1\}$, $a \in A$ e $1 \leq t \leq T$ con valore 1 se l'azione $a$ viene eseguita al tempo $t$, 0 altrimenti.

- $y^c_{f,g,t}$ , $c \in C,\ f, g \in D_c,\ 1 \leq t \leq T$ con valore 1 se la variabile di stato $c$ passa dal valore $f$ al valore $g$ al tempo $t$, 0 altrimenti.

Per comprendere la natura del modello proposto è bene prima illustrarne intuitivamente il funzionamento. Si immagini per ciascuna variabile di stato un grafo a livelli in cui ciascun livello sia composto da tutti i valori possibili in $D_c$ (vedi fig. 1.5).

[3] si noti che non è concesso usare un valore negativo come costo di una azione

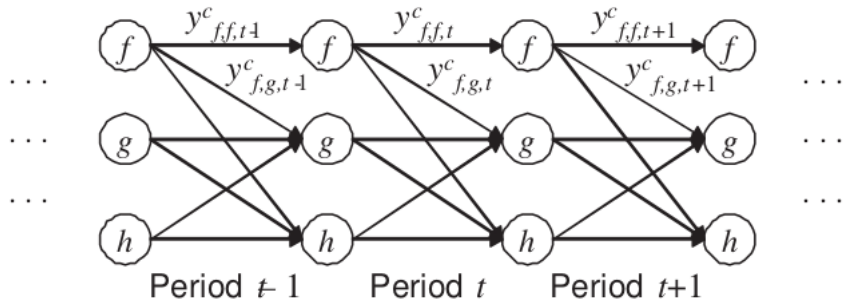


**Figura 1.5:** Esempio di flow graph [5]

Tali livelli saranno completamente connessi ciascuno con i suoi livelli direttamente prossimi (sia precedente che successivo). Ogni livello *i-esimo* rappresenterà il valore della variabile di stato al tempo $t = i$ e le connessioni tra un livello e il successivo sono invece i possibili cambiamenti di stato che potrebbe intraprendere la variabile. È facile notare come si crei una corrispondenza fra gli archi e le variabili $y^c_{f,g,t}$ prima definite.
Introducendo ora il modello vero e proprio:

$$min \sum_{t=1}^{T} \sum_{a \in A} cost_a \, x_{a,t} \tag{1.3}$$

$$\sum_{g \in D_c} y^c_{f,g,1} = \begin{cases} 1 & se \, f \in I_c \\ 0 & altrimenti \end{cases} \qquad \forall c \in C, g \in D_c \tag{1.4}$$

$$\sum_{h \in D_c} y^c_{g,h,t+1} = \sum_{f \in D_c} y^c_{f,g,t} \qquad \forall c \in C, \forall g \in D_c, 1 \leq t \leq T-1 \tag{1.5}$$

$$\sum_{f \in D_c} y^c_{f,g,T} = 1 \qquad \forall c \in G, g \in G_c \tag{1.6}$$

$$\sum_{a \in A:(f,g) \in SC_a(c)} x_{a,t} = y^c_{f,g,t} \qquad \forall c \in C, \; f, g \in D_c, f \neq g, 1 \leq t \leq T \tag{1.7}$$

$$x_{a,t} \leq y^c_{f,f,t} \qquad \forall c \in C, f \in D_c, a \in A, (f,f) \in PR_a(c), 1 \leq t \leq T \tag{1.8}$$

**Funzione obiettivo** (1.3)

L'equazione 1.3 definisce l'obiettivo del problema: utilizzare meno risorse possibili per arrivare allo stato goal. Si intende quindi minimizzare il costo totale del piano inteso come la somma di tutti i costi delle azioni utilizzate.

**Stato Iniziale** (1.4)

Il vincolo 1.4 impone i vincoli di stato iniziale affinché la somma degli archi uscenti da un nodo al tempo $t = 1$ sia: 0 se il valore non è quello definito allo stato iniziale $I$ oppure 1 altrimenti. Così facendo non vi è alcuna possibilità di propagazione di cambiamenti di stato se non dall'unico valore iniziale ammesso.

**Propagazione dello stato** (1.5)

L'espressione 1.5 è necessaria affinché i cambiamenti di stato delle variabili si propaghino correttamente da uno step al successivo. Infatti, la somma degli archi uscenti di un certo valore $g$ al tempo $t + 1$ deve essere pari a quella degli archi entranti in $g$ al tempo $t$.

**Goal Finale** (1.6)

Il vincolo 1.6 rappresenta la realizzazione dello stato di goal finale espresso in $G$. Per ogni variabile $c$ con un vincolo di goal $G_c$ viene infatti posto che la somma degli archi entranti sul valore goal al tempo $T$ debba essere uno, precludendo quindi implicitamente alcun altro valore possibile per via dell'equazione 1.5.

**Vincoli di State Change** (1.7)

Al punto 1.7 viene introdotto il collegamento fra i cambiamenti di stato $y^c_{f,g,t}$ e le variabili d'azione $x_{a,t}$ che abbiano un effetto di state change sulla variabile $c$. Si noti infatti come dato un tempo $t$, la somma di tutte le azioni con effetto $f \to g$ sulla variabile $c$ debba essere pari alla variabile di cambio stato $y^c_{f,g,t}$. Questo vincolo permette di introdurre un duplice significato:

- Dovesse venire usata una azione con effetto $f \to g$ allora l'arco del cambio di stato corrispondente, rappresentato da $y^c_{f,g,t}$, dovrà essere posto ad 1
- È possibile compiere al più una, e una sola, azione $a$ che introduca un cambiamento di stato $f \to g$ nella variabile $c$. Inoltre risulta impossibile per l'equazione 1.5 avere azioni di state change conflittuali ad uno stesso istante $t$.

**Vincoli di State Prevail** (1.8)

Concludendo il modello, vengono infine introdotti i vincoli di state prevail sulle variabili. Seguendo un ragionamento affine ai vincoli di state change, viene imposto come lower bound alle variabili di state prevail ($y^c_{f,f,t}$) il valore $x_{a,t}$ dove $(f, f) \in PR_a(c)$ cosicché vi fossero una o più azioni di state prevail anche l'arco di

riferimento dovrà rispecchiare l'immutabilità del valore della variabile $c$ al tempo $t$. Si noti come, a differenza degli state change, non vi sia un limite a quante variabili contemporaneamente possano avere effetti di state prevail.

Al fine di completare il modello è bene soffermarsi su casi piuttosto comuni ma che sfuggono a quanto proposto finora. Come gestire una azione che non ha pretese sul valore iniziale di una variabile ma ne impone uno di effetto? E se viceversa richiede un valore iniziale ma non ne specifica uno finale? Risulta quindi necessario definirne la gestione a seconda della casistica. Come si può notare dalla

| Caso | $c \in pre_a$ | $c \in eff_a$ | Commenti |
|---|---|---|---|
| 1 | $\emptyset$ | $\emptyset$ | Non necessario intervenire |
| 2 | $v$ | $\emptyset$ | Da definire |
| 3 | $v$ | $v$ | Standard Prevail |
| 4 | $v_1$ | $v_2(\neq v_1)$ | Standard State Change |
| 5 | $\emptyset$ | $v$ | Da definire |

**Tabella 1.1:** Riassunto esaustivo dei tipi di azione possibili

tabella 1.1 solo nelle casistiche 2 e 5 risulta necessario definirne il comportamento affinché siano compatibili con il modello. Nel caso 2 si può facilmente risolvere il problema rendendo l'azione uno standard prevail, ossia aggiungendo $eff_a(c) = v$ alla azione. Infatti è legittimo pensare che tale azione non abbia alcun interesse a modificare il valore e non presenta alcuno svantaggio poiché comunque viene concessa una sola azione di state change sulla variabile per istante di tempo. Il caso 5 risulta invece più complesso da gestire, suggerendo due differenti metodi per affrontarlo: **Transitional Normal Form** e **Multiply Actions**.

### 1.4.1 Transitional Normal Form

La Transitional Normal Form, d'ora in poi abbreviata in **TNF**, è una strategia basata sull'interpretazione del valore nullo in $eff_a(c)$.
Si procede ad estendere il dominio di tutte le variabili $c \in C$ aggiungendo un valore univoco $x^*$ che indichi lo stato indefinito della variabile.
Si procede inoltre alla creazione di nuove azioni $forget_{c,f}$ dal costo nullo, il cui effetto sia quello di trasformare la variabile $c$ da uno qualsiasi dei suoi valori $f \in D_c$ al valore univoco $x^*$. Si modificano infine tutte le azione di tipo 5 (vedi

1.1) affinché presentino il valore $x^*$ in $pre_a(c)$.

$$\begin{cases} D_c = D'_c \cup \{x^*\} \quad \forall c \in C \\ forget_{c,f} \coloneqq < c = f, c = x^*, 0 > \quad \forall c \in C, f \in (D_c \setminus \{x^*\}) \end{cases} \tag{1.9}$$

Il Planner potrà ora rendere la variabile $c$ indefinita senza costo alcuno ad un qualsiasi time step $t$ per poi utilizzare liberamente l'azione di tipo 5.
Tale approccio viene però ad un costo: l'utilizzo di variabili che di fatto rientrano negli state change dilunga gli step necessari alla risoluzione al più di un fattore 2. Questo genera un aumento sostanziale delle variabili di azione e stato associate al problem MILP.

### 1.4.2 Multiply Actions

Multiply Actions, come suggerisce il nome, è una strategia che elude il problema del valore indefinito e si concentra invece direttamente sulla creazione di nuove azioni a partire da quella anomala originale.
Nello specifico, partendo da una ipotetica azione di tipo 5 (vedi 1.1): $a^* \coloneqq < c = \emptyset, c = g, 1 >$ si procede a generare nuove azioni che abbiano come valore di partenza tutti i valori possibili di c, e come effetti e costo gli stessi di $a^*$, in maniera tale che il planner a seconda della situazione usi una diversa variante di $a^*$.

$$a^*_f \coloneqq < c = f \cup pre_{a^*}, eff_{a^*}, cost(a^*) > \quad \forall f \in D_c \tag{1.10}$$

Il Planner potrà quindi utilizzare l'azione $a^*_f$ compatibile con il valore di $c$ in quel momento.
Se questo approccio non aumenta l'orizzonte temporale, può comunque portare ad un aumento esponenziale delle variabili di azione a seconda della grandezza del dominio $D_c$ e della quantità di azioni così modificate.

# Capitolo 2

# Caso in esame

Il confronto effettuato in questo studio iniziale, si concentra sulle prestazioni di modelli di tipo **TNF** e **MA** rispetto ad algoritmi di ricerca euristici come $A^*$.

## 2.1 Istanze di benchmark

In questo studio preliminare vengono utilizzate 150 diverse istanze di problemi di AI Planning appartenenti alla classe MICONIC. I problemi sono stati scelti come mediamente semplici: essi infatti si compongono al più di 3600 operatori e 61 variabili di stato. Le istanze sono state numerate secondo una denominazione $(i, j) \quad i \in \{1, 2, .., 30\}, j \in \{0, 1, 2, 3, 4\}$. All'aumentare di $i$ o $j$ i problemi aumentano di dimensioni, permettendo quindi una analisi delle prestazioni in diverse situazioni di carico.
Poiché tipicamente descritti in PDDL [6], i problemi di AI Planning spesso non sono facilmente interpretabili a causa della grammatica inconstante di tale linguaggio e l'incompatibilità con sistemi moderni. Per rimediare al problema è stato utilizzato come parser *Fast Downward* [7] con alcuni caveat sul tipo di output utilizzabili[1].
Ogni problema è definito da variabili con dominio di tipo $D_c = \{0, 1, ..., N\}$ e le azioni sono state manipolate come precedentemente descritto (vedi 1.1) a seconda del metodo implementativo utilizzato.

[1]Vengono considerati non validi output che presentino axiom-layer $\neq 1$, effect conditions, axiom section non vuota o sezione mutex non vuota. Vedi [8] per ulteriori dettagli

## 2.2 PYOMO e CPLEX

Per quanto riguarda l'aspetto implementativo, si è realizzato il modello in *Python 3.8.10* con l'ausilio della libreria PYOMO[2]. PYOMO è una libreria open-source in Python atta alla creazione di modelli per problemi di ottimizzazione di vario tipo [9] tra cui la programmazione lineare intera mista.
Si è scelta questa combinazione per la estrema facilità di utilizzo di Python per il parsing delle istanze, oltre alla ricca documentazione e community da cui è contraddistinto PYOMO stesso. È importante chiarire come Pyomo funga esclusivamente da creatore del modello e, di per sè, sia sprovvisto di alcuna funzionalità di diretta risoluzione. Nonostante possa sembrare contradditorio è invece una scelta saggiamente ponderata: infatti i risolutori sono elementi tanto potenti quanto complicati, e necessitano di grande attenzione e cura per poter essere competitivi. A seconda dell'approccio utilizzato dalla sua casa madre, ogni risolutore si presta quindi in maniera leggermente diversa a differenti problemi. In questo esperimento è stato utilizzato CPLEX[3]. CPLEX è un risolutore esclusivamente pensato per problemi di programmazione lineare intera [10], capace di interfacciarsi con Pyomo ed interpretarne i modelli espressi.
La combinazione scelta (PYOMO + CPLEX) permette quindi di avere la flessibilità tipica di Python ma con l'efficienza di calcolo di CPLEX, in maniera tale da avere un riferimento competitivo rispetto al mercato.

## 2.3 Conduzione dell'esperimento

La comparazione di performance è stata effettuata sulla medesima macchina, sia per l'algoritmo $A^*$ che per le formulazioni MIP TNF/MA. Sarà chiaro che vi saranno dei bias di cui tenere conto nell'analisi dei risultati, si è scelto per praticità di spostare queste riflessioni nel capitolo 3.
L'esperimento si compone di tre fasi principali:

- La risoluzione delle istanze con FastDownward [7], un planner allo stato dell'arte attuale con strategia $A^*$
- La risoluzione delle istanze tramite formulazione MIP con approccio TNF
- La risoluzione delle istanze tramite formulazione MIP con approccio MA

[2] `https://pyomo.readthedocs.io/en/stable/`
[3] `https://www.ibm.com/analytics/cplex-optimizer`

Per evitare bias dovuti alla potenza di calcolo è stata utilizzata sempre la stessa macchina (o una ad essa equivalente) per l'esecuzione di tutte le istanze sopracitate.
Le metriche di interesse valutate sono:

- **tempo di risoluzione**, inteso come il tempo effettivo trascorso da ciascun approccio alla risoluzione di una stessa istanza. Il tempo è misurato secondo lo scorrere reale, partendo da quando il risolutore comincia a lavorare sull'istanza.

- **gap primali e duali** alla fine del nodo radice nel caso di approcci TNF e MA. Questo per permettere una migliore analisi dei punti possibilmente critici dei modelli.

Nello specifico, dapprima si è utilizzato il Planner $A^*$ per risolvere le istanze. Questo ha permesso di avere la quasi totalità delle soluzioni con conseguente orizzonte temporale del problema stesso. Per la formulazione *1SC* è infatti necessario specificare un tempo $T$ di risoluzione massimo. Poiché la dimensione del problema cresce sensibilmente all'aumentare di $T$, si è quindi deciso di utilizzare nei modelli MIP un valore di $T$ già precedentemente validato dal Planner $A^*$.
È bene ricordarsi che la formulazione TNF può al più raddoppiare il tempo di esecuzione a causa delle azioni di forget, si è quindi provveduto a fornire un tempo $T = 2T^*$ alle istanze TNF, dove $T^*$ è il numero di step necessari per risolvere il problema per il Planner $A^*$.
Tutti e tre i metodi hanno ricevuto un tempo massimale di 1 ora per la risoluzione dell'istanza. In caso tale tempistica venga ecceduta si assegna un tempo di risoluzione pari a 3600 secondi a prescindere dal tempo ancora necessario ad ottenere una soluzione.
Una volta estrapolati i dati di risoluzione e i log di CPLEX[4] si è proceduto all'analisi di tali dati. Sono stati quindi calcolati per ciascuno dei planner:

- La percentuale di istanze risolte in funzione del tempo

- La media geometrica del tempo di risoluzione di ciascun metodo

- La media geometrica del tempo di risoluzione per le seguenti fasce di tempo (espresse in secondi): [0, 60[, [60, 600[, [600, 3601[[5]

[4] necessari all'analisi di gap primali e duali
[5] esclusivamente per i metodi TNF, MA

- I gap primali e duali dei metodi TNF e MA alla fine del nodo radice

Laddove ritenuto necessario sono stati rappresentati i risultati numerici con grafici comparativi.

### 2.3.1 Ulteriori risorse utilizzate

Oltre all'utilizzo di svariate funzioni della liberia standard di Python, sono stati inoltre utilizzate le librerie MatPlotLib [11], Pandas [12] e NumPy [13]. Tali packages sono stati principalmente impiegati per la realizzazione dei grafici e saltuariamente per la gestione di calcoli proni all'overflow.

# Capitolo 3

# Risultati Sperimentali

In questo capitolo vengono presentati i risultati empirici collezionati dagli esperimenti fatti. È importante affrontare in ordine il capitolo perché permette una chiara e precisa prima interpretazione dei risultati mostrati.

**Analisi percentuali istanze risolte**

Questo tipo di analisi è stata condotta osservando in base al metodo quante istanze fosse in grado di risolvere entro il time limit di un'ora. I dati così riportati sono calcolati come nell'equazione 3.1 e relazionati in funzione del tempo.

$$\frac{istanze\ risolte}{150\ istanze\ totali} \tag{3.1}$$

**GAP primale e duale**

Il GAP primale e quello duale sono stati calcolati rispettivamente con le equazioni 3.2, 3.3 con i valori trovati alla fine dell'analisi del nodo radice. Nell'eventualità tali valori non fossero presenti, si è assegnato il valore zero, facendo quindi convergere il GAP a valore 1.

$$\frac{miglior\ incumbent\ -\ soluzione\ effettiva}{soluzione\ effettiva} \tag{3.2}$$

$$\frac{rilassamento\ più\ stringente\ -\ soluzione\ effettiva}{soluzione\ effettiva} \tag{3.3}$$

**Media geometrica con shift**

Per tale calcolo si è considerata la media geometrica dei tempi di risoluzione con

uno shift $s = 1sec$ applicato sia prima che dopo la media (vedi equazione 3.4).

$$\sqrt[150]{\prod_{i=1}^{150}(t_i^* + s)} - s \tag{3.4}$$

Inoltre sono state calcolate anche le medie geometriche su insiemi ristretti per quanto riguarda TNF e MA (vedi capitolo 2 per appronfondimenti).

## 3.1 Definendo i Bias

Come risulterà chiaro, la conduzione dei confronti scelta favorisce fortemente i metodi TNF e MA. Questa tendenza non è stata eliminata principalmente per due motivi:

1. Il lavoro qui presente è una *proof of concept*, non vi è la necessità di competere effettivamente con le soluzioni commerciali ma piuttosto di dimostrare che sia una strada più o meno viabile. Pertanto essendo uno studio preliminare non è data poi una così grande importanza ai risultati numerici se non come indici generici ed approssimativi.

2. Per alcuni bias affrontati vi è una effettiva motivazione pratica o reale per cui è lecito pensare di avere tali vantaggi.

### 3.1.1 Bias orizzonte temporale

Questo bias è principalmente dovuto all'utilizzo del tempo $T$ recuperato dal Planner $A^*$. Risulta necessario infatti prima risolvere il problema con un mezzo esterno per poter utilizzare poi la modellazione MIP, vanificandone però completamente il senso poiché una soluzione risulta già trovata dal Planner $A^*$.
Chiaramente il vantaggio portato ai metodi TNF e MA non è da poco, ciononostante è lecito pensare ad approcci misti in cui si utilizzi un primo metodo euristico ma non ottimo[1] dalle prestazioni estremamente competitive e, solo successivamente, una soluzione MIP per affinare il risultato grezzo precedentemente trovato. Per questo motivo il bias va tenuto in considerazione ma risulta parzialmente giustificato nell'ottica di utilizzo reale di una di queste tecniche.

[1] inteso come incapace di provare l'ottimalità della sua soluzione

### 3.1.2 Bias time limit

Imporre un time limit sull'esecuzione del programma favorisce fortemente i metodi più lenti perché permette loro di abbassare considerevolmente le loro medie di risoluzione. In questo caso non vi è una situazione reale per cui si possa giustificare tale comportamento.
Questa scelta è stata effettuata a posteriori dopo una analisi senza time limit. Ci si è resi conto che le performance del Planner $A^*$ erano talmente superiori a TNF e MA che introdurre un time limit era necessario principalmente per ridurre i tempi di esecuzione delle 150 istanze e delineare un confine marcato per cui si possa ritenere "oltre il beneficio del dubbio" la viabilità di approcci di tipo MIP. Ciò nonostante, è importante non scordarsi di tale vantaggio nelle medie, che risultano quindi particolarmente favoreggianti gli approcci di tipo MIP.

### 3.1.3 Bias gap su instanze risolte

Il gap duale e primale viene calcolato sulle istanze per cui il metodo sia perlomeno riuscito ad approntare un tentativo di soluzione. Con "tentativo" si intende che il risolutore CPLEX abbia cominciato ad analizzare l'albero B&B del problema, generando quindi incumbent e valori di rilassamento per l'istanza in considerazione.
Tuttavia, nel momento in cui il risolutore non riesce nemmeno ad eseguire questi primi calcoli, l'istanza non viene considerata ai fini dell'analisi dei dati.
Questo fenomeno può potenzialmente favorire i metodi meno performanti che, riuscendo ad affrontare solo le istanze più semplici, godrebbero di gap più stringenti dovuti però alla facilità del problema più che al loro potere espressivo.
È dunque necessario tenere in considerazione anche il numero di istanze risolte da ciascun approccio per evitare di essere fuorviati da gap primali e duali efficaci ma presenti in esiguo numero.

## 3.2 Risultati Numerici

**Percentili di risoluzione**
Considerando un time limit di un'ora, i diversi approcci sono riusciti a risolvere:

- MA: il 57.9% delle istanze
- TNF: il 46.6% delle istanze

- Planner $A^*$: il 94.6% delle istanze

**GAP primale/duale**

La media aritmetica dei GAP trovati con MA e TNF è stata:

- MA: 27.49% gap primale, 1.56% gap duale

- TNF: 10.43% gap primale, 2.71% gap duale

**Media Geometrica**

Considerando un time limit di 3600 secondi, la media geometrica è stata calcolata con uno shift di 1 secondo (vedi 3.4) su tutte le 150 istanze. I risultati cosi ottenuti sono stati:

- MA: 207.97 secondi

- TNF: 410.97 secondi

- Planner $A^*$: 2.47 secondi

Per quanto riguarda le analisi in tempi diverse fasce di risoluzione effettuate su TNF e MA si rimanda alla tabella sottostante.

| Fascia di tempo (secondi) | MA | TNF |
|---|---|---|
| $[0, 60[$ | 1.18 sec | 2.64 sec |
| $[60, 600[$ | 120.06 sec | 319.51 sec |
| $[600, 3600[$ | 3046.99 sec | 3600.00 sec |

## 3.3 Grafici

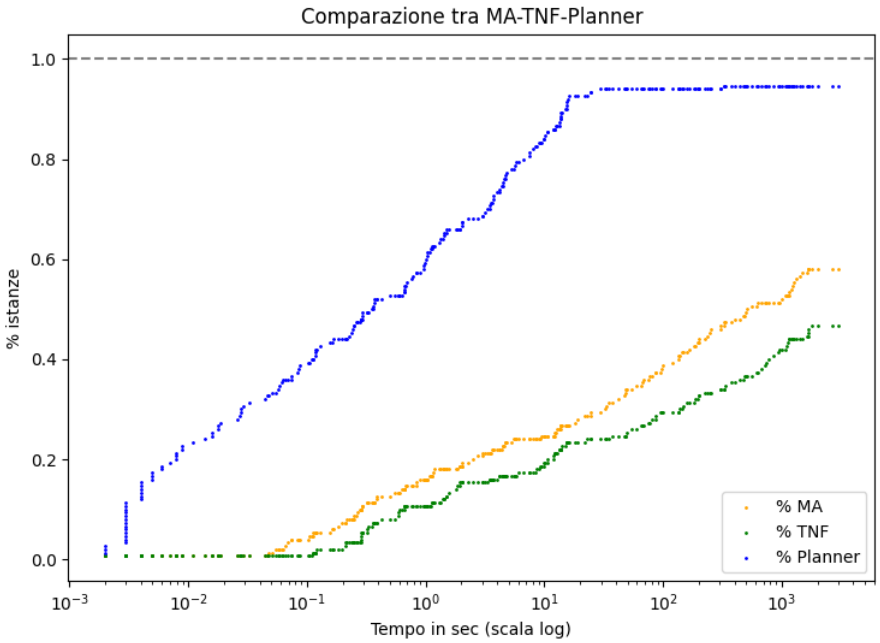


**Figura 3.1:** Percentuali di istanze risolte in funzione del tempo di tutti i metodi affrontati ($A^*$ in blu, TNF in verde, MA in arancio)

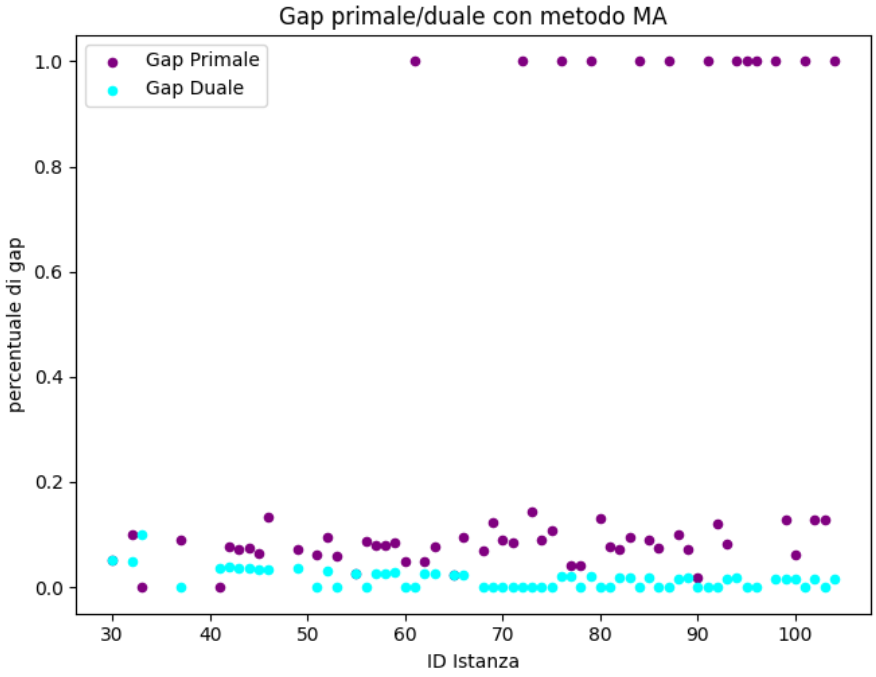


**Figura 3.2:** Percentuali di GAP duale e primale - MA

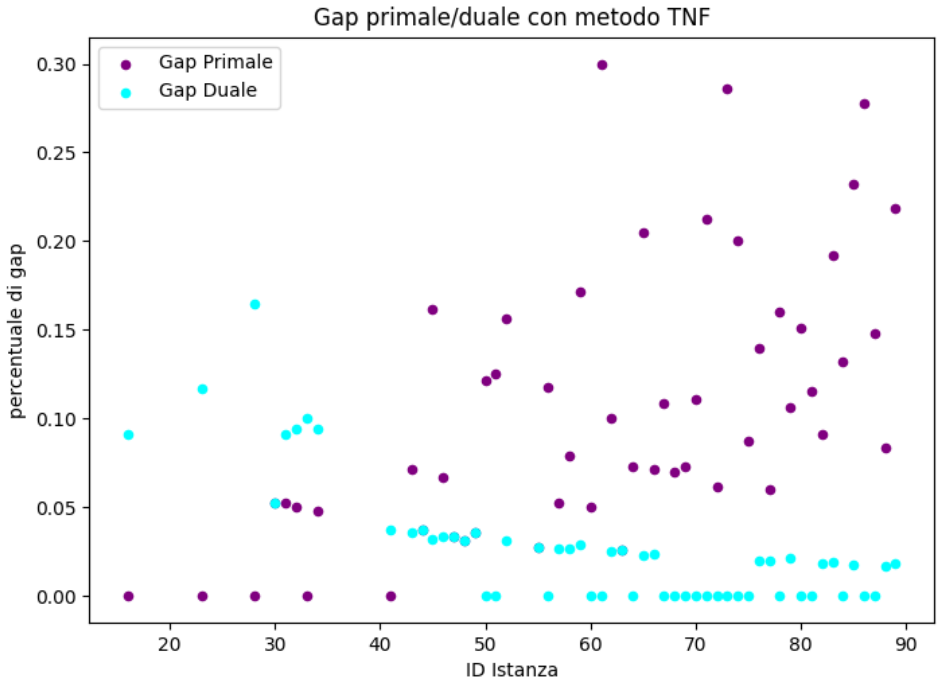


**Figura 3.3:** Percentuali di GAP duale e primale - TNF

# Capitolo 4

# Analisi e Conclusioni

Questo capitolo è dedito all'analisi dei dati rappresentati al capitolo 3. Si ricorda che è importante inoltre avere compreso i bias introdotti al capitolo 2 affinché risultino più chiare le considerazioni fatte.

## 4.1 Analisi Performance

Risulta chiaro da tutti gli indici analizzati come il Planner $A^*$ sia nettamente superiore agli approcci MIP. Sottolineandone i punti di forza si può osservare come:

- Il Planner $A^*$ abbia risolto la quasi totalità delle istanze (94.6%), circa il doppio delle istanze risolte da MA e TNF.
- Nonostante abbia affrontato istanze più difficili, ha comunque mantenuto una media geometrica[1] due ordini di grandezza inferiore agli approcci MIP.

Nonostante i vantaggi, MA e TNF hanno performato considerevolmente peggio rispetto ad $A^*$. Si osservi inoltre la figura 3.1 per realizzare come non solo abbia risolto più istanze di MA e TNF, ma FastDownward abbia inoltre raggiunto il suo limite superiore considerevolmente prima. In sintesi, gli approcci MIP non hanno retto confronto con lo stato dell'arte.
Volendo invece comparare tra loro gli approcci MA e TNF l'analisi si fa invece più dettagliata. Fermo restando che Multiply Actions sembra l'approccio migliore per questo tipo di istanze, i due metodi hanno risolto le istanze nel medesimo range di grandezza.

[1] con shifting di un secondo

Osservando con più attenzione però, si notano vantaggi sostanziali da parte di MA: ha impiegato mediamente il 50.5% del tempo in meno rispetto TNF. Inoltre, tale differenza si fa ancora più marcata se rapportata alle fasce temporali $[0, 60[$ e $[60, 600[$.
Tuttavia il vantaggio non si trasferisce altrettanto facilmente nella potenza della formulazione del modello. Infatti TNF presenta un GAP duale comparabile con MA, ma un GAP primale più che dimezzato in media. Tali risultati conflittuali sono riconducibili al minor numero di istanze risolte da parte di TNF.
Avendo affrontato istanze più semplici ha avuto il tempo di trovare incumbent effettivi laddove invece MA per dimensione del problema non è riuscito entro la fine del nodo radice. Tale comportamento è ben evidenziato nella figura 3.2 dove si nota come solo con l'avanzare delle istanze, e quindi l'aumento della dimensionalità, MA cominci a fallire nella ricerca di un incumbent nelle fasi iniziali di risoluzione. Inoltre il grafico 3.3 rivela un andamento più scostante nel tempo, con svariati picchi del 30% circa nei gap primali.
Generalmente vi è comunque un trend in entrambi i grafici per cui il gap primale risulta considerevolmente più alto di quello duale, tale comportamento è però inusuale per i problemi di tipo MIP.
Solitamente infatti, la difficoltà del risolutore risiede nel provare che l'incumbent trovato sia di fatto la soluzione migliore. Quanto osservato in questi modelli invece, suggerisce il contrario: il risolutore trova fin da subito un lower bound dal valore alto, ma fatica a trovare soluzioni intere di qualità.
Tale problema potrebbe essere arginato usando soluzioni (anche non ottimali) al problema come *upper cut off* per il risolutore, ossia dando in input un valore che si sa già essere soluzione valida al problema a CPLEX in maniera tale che possa scartare l'analisi di tutti i sotto alberi con possibili soluzioni peggiori dell'attuale upper-bound propostogli.

## 4.2 Possibili Miglioramenti e Approfondimenti

La modellazione proposta (1 SC) è una delle più basiche tra quelle proposte nel paper di riferimento [5]. Vi è sicuramente spazio per migliorare nella ricerca di formulazioni dinamiche e più stringenti.
Per migliorare i risultati empirici dell'esperimento si potrebbe anche dare come orizzonte temporale $T^*$ ai modelli un valore non per forza ottimale, ed osservare se MA soffra dell'aumento di dimensionalità più di quanto faccia TNF.

Non è inoltre da sottovalutare la possibilità che approcci di tipo MIP permettano una ricerca più efficiente in caso di problemi di AI Planning dalle dimensioni ben più considerevoli. Tuttavia tale possibilità è comunque da valutare poiché i modelli finora osservati soffrono particolarmente il *curse of dimensionality*.

## 4.3 Conclusioni

Con il modello affrontato in questa tesi è chiaro che la strada MIP non sia una soluzione ne competitiva ne viabile. Tuttavia vi è sicuramente spazio per lo studio di formulazioni più avanzate dal potere discriminante più forte. Tali modelli potrebbero risultare comparabili ad approcci euristici, in quanto utilizzano tecniche più sofisticate per la gestione dei vincoli.
Un'ulteriore spunto di miglioramento potrebbe essere tratto dall'utilizzo dell'upper cut off come suggerito al sottoparagrafo precedente.
All'interno degli approcci MIP affrontati, MA si è dimostrato considerevolmente più efficace. Tuttavia, non è da escludersi che sia la natura delle istanze di benchmark a favorire tale tipo di soluzione. In conclusione, per problemi dalle dimensioni contenute Multiply Actions risulta preferibile alla Transitional Normal Form.

# Bibliografia


[1] *AI Planning - IBM*, https://researcher.watson.ibm.com/researcher/view_group.php?id=8432, ultima consultazione 15/06/2022.

[2] Stuart Russell, Peter Norvig *Artificial Intelligence: A Modern Approach*, 2015.

[3] *Mixed Integer Programming Basics*, https://www.gurobi.com/resource/mip-basics/, ultima consultazione: 16/06/2022.

[4] J. Dongarra, F. Sullivan *Guest Editors Introduction to the top 10 algorithms* https://www.computer.org/csdl/magazine/cs/2000/01/c1022/13rRUxBJhBm 2000.

[5] Menkes van den Briel, Thomas Vossen, Subbarao Kambhampati *Reviving Integer Programming Approaches for AI Planning: A Branch-and-Cut Framework.* https://www.researchgate.net/publication/220935968_Reviving_Integer_Programming_Approaches_for_AI_Planning_A_Branch-and-Cut_Framework

[6] *What is PDDL?* https://planning.wiki/guide/whatis/pddl, ultiman consultazione 20/06/2022.

[7] *Fast Downward - Homepage* https://www.fast-downward.org/, ultima consultazione 20/06/2022.

[8] *Fast Downward - OutPut Format* https://www.fast-downward.org/TranslatorOutputFormat, ultima consultazione 20/06/2022.

[9] *Pyomo - About* https://pyomo.readthedocs.io/en/stable/#, ultima consultazione 20/06/2022.

[10] *What is CPLEX?* https://www.ibm.com/docs/en/icos/20.1.0?topic=mc-what-is-cplex, ultima consultazione 20/06/2022.

[11] *MatPlotLib - Hompage* `https://matplotlib.org/`, ultima consultazione: 21/06/2022.

[12] *Pandas - Homepage* `https://pandas.pydata.org/`, ultima consultazione 21/02/2022.

[13] *NumPy - Homepage* `https://numpy.org/`, ultima consultazione 21/02/2022.