UNIVERSITÀ DEGLI STUDI DI TORINO

SCUOLA DI SCIENZE DELLA NATURA

Corso di Laurea in Informatica

Tesi di Laurea

Un approccio mixed-initiative per la pianificazione con vincoli su risorse consumabili

Relatore:

Prof. Pietro Torasso

Co-relatore:

Dott. Enrico Scala

Candidato:

Davide Dell'Anna

Anno Accademico 2012/2013

Ringraziamenti

Desidero ringraziare innanzitutto il Prof. Pietro Torasso per i preziosi insegnamenti forniti durante il corso di studio e la straordinaria disponibilità dimostratami durante lo sviluppo del progetto e la scrittura della tesi.

Ringrazio inoltre il Dott. Enrico Scala per avermi fornito preziosi consigli e il supporto necessario per la realizzazione del progetto.

Infine un ringraziamento speciale va ai miei genitori, che mi hanno dato l'opportunità di intraprendere questo percorso di studi e di concluderlo nel migliore dei modi.

Indice

Capitolo 1

Introduzione

La pianificazione è una componente particolarmente importante dell'intelligenza artificiale. Con questo termine si intende l'elaborazione di un piano di azione per raggiungere un determinato obiettivo.

Un problema di pianificazione è dunque un problema di ricerca di una sequenza di azioni che, rispettando i vincoli imposti dal dominio del problema, porti al raggiungimento di un obiettivo. Problemi di questo tipo si compongono di tre elementi: uno stato iniziale (la situazione di partenza), uno stato finale (la situazione in cui vorremmo arrivare, l'obiettivo da raggiungere) e un insieme di azioni che, quando applicate, provocano un cambiamento di stato.

Una azione può essere o meno applicabile in un certo stato e, se applicata, ha delle conseguenze sugli elementi del dominio e porta in un altro stato. Gli stati, all'interno del problema, sono delle descrizioni ad alto livello simbolico del dominio in cui stiamo operando e ne rispettano i vincoli. Le azioni consistono principalmente in un insieme di precondizioni e in un insieme di effetti. Le prime sono dei vincoli che definiscono se una azione può essere applicata o meno in un certo stato (in base ai valori delle variabili in tale stato), i secondi rappresentano il risultato dell'esecuzione di una azione in termini di cosa cambia all'interno del dominio. Risolvere un problema di ricerca vuol dire chiedersi se esiste una sequenza ordinata di azioni che porti dallo stato iniziale a quello finale.

La pianificazione, così come descritta fino ad ora, è detta classica o proposizionale ([8]), ed è in grado di trattare problemi non particolarmente complessi, nei

quali c'è la necessità di capire cosa fare e in quale ordine farlo, ma non c'è bisogno di tener conto di fattori che durante l'esecuzione di un piano possono variare, quali il tempo o il consumo di risorse. Fino a qualche anno fa, per via delle limitate capacità tecnologiche, la ricerca sulla pianificazione si è limitata quasi esclusivamente a problemi di questo tipo.

Negli ultimi anni, invece, ci si è avvicinati sempre più alla pianificazione di problemi realistici. Quando si ha a che fare con problemi reali complessi, come ad esempio la pianificazione di una spedizione di un rover nello spazio, sorge la necessità di gestire anche elementi variabili come quelli sopracitati. Ogni azione che deve essere eseguita, infatti, ha una durata temporale e consuma una determinata quantità di risorse.

Tali risorse spesso sono consumabili e finite. È il caso ad esempio della quantità di alloggiamenti disponibili su un rover per prelevare campioni di suolo. Talvolta le risorse consumabili possono essere riutilizzabili, ovvero inizialmente sono disponibili in quantità finita, nel corso dell'esecuzione possono venire consumate, ma dopo un'operazione di "ripristino" possono tornare ad essere utilizzabili (e.g. la memoria a disposizione su un rover è limitata e può essere riempita completamente, tuttavia dopo eventuali operazioni di trasmissione a terra dei dati catturati essa viene svuotata e resa nuovamente utilizzabile). Analogamente le risorse consumabili possono essere rinnovabili. In tal caso la risorsa già utilizzata non è strettamente riutilizzabile, però può venirne prodotta una nuova quantità (e.g il *power* di un rover una volta consumato non è riutilizzabile, tuttavia poiché può provenire da pannelli fotovoltaici, può essere rigenerato).

In questi casi, perciò, occorre effettuare una pianificazione che tenga conto di tali vincoli, associando ad ogni azione il corrispondente tempo necessario per eseguirla e la quantità di risorse consumate.

Soluzioni a problemi di questo tipo devono dunque soddisfare, oltre ai vincoli proposizionali, anche i vincoli numerici riguardanti le risorse consumabili. Inoltre devono tener conto dei vincoli numerici di ordine temporale, i quali possono essere associati sia alla durata del piano in sè, sia allo stesso utilizzo delle risorse, in relazione alla loro disponibilità rinnovabile/riutilizzabile o meno. Per esempio la necessità di scattare una fotografia durante una missione di esplorazione può dipendere dal fatto che sia in corso un evento specifico. In tal caso il piano deve

rispettare dei vincoli temporali relativi a quel determinato evento e dunque le risorse necessarie per l'esecuzione dell'azione devono essere disponibili in un preciso intervallo di tempo. Allo stesso tempo può esservi l'esigenza di eseguire l'intero piano entro una determinata quantità di tempo, pertanto l'insieme delle azioni da eseguire dovrà rispettare tali vincoli.

Questi accenni lasciano intuire quanto la pianificazione con vincoli su risorse consumabili sia tutt'altro che banale e le ragioni per le quali essa sia stata affrontata solamente in un tempo recente.

Lo sviluppo della pianificazione con vincoli di risorse è stato favorito, in particolare, dal rilascio di PDDL 2.1 [5] (evoluzione di PDDL, il linguaggio diventato lo standard per la rappresentazione di modelli di dominio), il quale fornisce una notevole potenza di modellazione. Esso è infatti in grado di modellare classi di domini che richiedono una pianificazione dell'uso delle risorse e del tempo, tramite l'uso di fluenti numerici, permettendo di dividere, all'interno del problema, la parte predicativa da quella numerica.

Per realizzare un piano, con il tempo si sono tentati principalmente due approcci: un approccio manuale e uno automatico.

La pianificazione manuale prevede la costruzione del piano da parte di un operatore umano. Il risultato di una operazione di questo tipo dipende molto dalle capacità e dall'esperienza dell'operatore nell'ordinare i compiti rispettando i vincoli del problema. Naturalmente l'utilizzo di questo approccio risulta molto complesso e costoso in termini di tempo, ed è possibile solamente per problemi non particolarmente articolati, in cui il numero di variabili e di vincoli è piccolo, e per problemi riguardanti ambienti completamente deterministici e prevedibili. Inoltre, anche riuscendo a trovare un piano, è molto improbabile che esso sia ottimale, per via dell'enorme quantità di fattori di cui l'operatore deve tener conto.

La pianificazione automatica, al contrario, delega il compito di costruire il piano interamente al sistema intelligente. Questo approccio permette di risparmiare molto tempo rispetto a quello manuale e può garantire una soluzione ottima. Non si può però considerare come una tecnica perfetta. È possibile infatti che una soluzione trovata dal pianificatore automatico, non sia poi effettivamente applicabile in quanto il sistema può non essere a conoscenza di informazioni non fornitegli perché impossibili da esprimere. Inoltre, anche la pianificazione automatica può

funzionare solamente con domini deterministici e prevedibili. Quando si trattano problemi reali, tuttavia, si ha a che fare con ambienti molto dinamici e scarsamente prevedibili, di cui si ha spesso una conoscenza solo parziale e in cui c'è la possibilità che si verifichino, a tempo di esecuzione, degli inconvenienti inaspettati che possono andare a modificare il mondo, rendendo inconsistente il piano realizzato in precedenza.

Per queste ragioni, negli ultimi anni si è intrapresa la strada della Mixed Initiative Planning (si vedano ad esempio [2], [3], [4]). Questo nuovo approccio prevede una "collaborazione" tra il sistema intelligente e l'operatore umano. Quest'ultimo infatti, dopo aver fornito al sistema lo stato iniziale e gli obiettivi, può delegargli una prima pianificazione, per poi analizzarne il risultato e decidere cosa modificare, per raggiungere un grado di ottimalità maggiore, eventualmente variando anche i *goals* ([3]). Oppure può decidere di costruire un po' alla volta un piano con il supporto del sistema, il quale garantisce la consistenza del piano stesso ([2]). In questo modo si risparmia la grande quantità di tempo che nella pianificazione manuale si occupava per costruire il piano, e allo stesso tempo si gestisce il problema della rigidità della pianificazione automatica permettendo all'operatore di agire sul piano costruito dal sistema. Tutto ciò dunque permette di affrontare situazioni che prima non venivano considerate. Grazie alla possibilità di modificare valori del piano, si possono infatti esaminare più agevolmente delle alternative e simulare il verificarsi di eventi inaspettati.

Una delle ragioni per cui solamente negli ultimi anni ci si è avvicinati alla Mixed Initiative, è il fatto che si sia sempre data maggior importanza all'ottimizzazione della pianificazione piuttosto che all'esecuzione di un piano. Concentrandosi su come risolvere i problemi di pianificazione, sono stati trascurati fattori quali la possibilità di contingenze a tempo di esecuzione e l'eventuale necessità di modificare un piano già costruito. Situazioni di questo tipo, tuttavia, come già detto, sono molto frequenti in ambienti reali, nei quali non è possibile prevedere con certezza tutto ciò che succederà. In tali ambienti non è sufficiente utilizzare un piano completamente fissato in precedenza o soluzioni come piani condizionali costruiti a tempo di pianificazione (piani in cui tutte le possibili contingenze che si pensa possano occorrere durante l'esecuzione vengono rappresentate come delle strade alternative che l'agente potrà intraprendere [8]). Queste soluzioni, infatti,

non sono abbastanza robuste in ambienti, come un pianeta da esplorare, in cui è impossibile prevedere ogni evenienza.

Un approccio misto, associato ad adeguate tecniche di ripianificazione o riconfigurazione online, permette invece di sviluppare, offline, un piano per il quale, analizzando tutti gli inconvenienti prevedibili, si può scegliere una configurazione abbastanza versatile da essere in grado di gestire un adeguato numero di contingenze. A tempo di esecuzione, poi, nel caso tale piano venga reso inconsistente, si potranno sfruttare le capacità computazionali dell'agente per modificarlo.

Lo scopo di questa tesi è lo sviluppo di un sistema grafico che dia supporto all'utente durante la fase di pianificazione. Il sistema fornirà all'operatore umano strumenti che gli permettano di capire facilmente in che condizioni si trova il piano e gli permettano di modificarlo attraverso funzioni di controllo, alterazione ed esecuzione di ciò che è stato realizzato. L'obiettivo è permettere un approccio misto nella realizzazione di un piano, ossia permettere una pianificazione interattiva che consenta all'operatore sia di costruire manualmente un piano o di effettuare le modifiche che ritiene opportune, sia di richiamare funzioni automatiche del sistema che vanno dal controllo di esistenza di un piano alla richiesta di configurazione di un piano secondo certi criteri. L'operatore avrà quindi la possibilità di costruire graficamente un piano da zero, scegliendo quali azioni inserire, modificare o eliminare, e quali modalità di esecuzione (vedi capitolo 3) associare ad esse. Potrà inoltre chiedere al sistema di verificare la validità del piano costruito, di ottimizzarlo o di simularne l'esecuzione.

Il progetto, inoltre, è una estensione del lavoro sulle MMA (Multi Modality Actions) del Dott. Scala in [9], e ne richiama ed integra gli strumenti realizzati. Il sistema, che è indipendente dal dominio su cui si vuole lavorare, permette infatti di realizzare, in maniera più intuitiva e user-friendly, piani con MMA, che si ritengono essere una scelta opportuna in casi in cui ci sia bisogno di tener conto di risorse numerabili e di eventuali inconvenienti non prevedibili a tempo di pianificazione. Le Multi Modality Actions (vedi capitolo 3) consentono infatti di definire, per ogni azione del piano, più modalità di esecuzione. Questo permette, ogniqualvolta sorga un imprevisto, di non dover effettuare necessariamente una completa ri-pianificazione, ma di tentare, inizialmente, solo una riconfigurazione

delle modalità scelte per le azioni. Un'operazione di questo tipo è più efficiente e meno costosa a livello computazionale rispetto ad una nuova pianificazione, e permette di mantenere inalterata la struttura generale del piano (non altera nè la sequenza nè l'insieme delle operazioni da eseguire) andando a modificare solamente le modalità con cui vengono eseguite le azioni.

La tesi è organizzata come segue:

- Nel capitolo 2 verrà fatta un'introduzione alla pianificazione con vincoli di risorse numeriche. Verranno descritti il processo di pianificazione e i principali linguaggi di rappresentazione di domini e problemi. Verranno inoltre presentate le principali caratteristiche di PDDL, il corrente linguaggio standard per la rappresentazione di domini. Infine verranno brevemente confrontati due pianificatori, il primo dei quali (FF) è indirizzato ad una pianificazione classica senza l'uso di risorse numeriche, mentre il secondo (Metric-FF) è orientato all'uso di variabili numeriche.

- Nel capitolo 3 verranno brevemente presentate le azioni multi-modalità, con riferimento al lavoro di Scala in [9], per permettere al lettore di capire meglio quanto sviluppato. Verrà illustrata la sintassi delle MMA, e il loro rapporto con PDDL e, infine, verrà descritta l'operazione di riconfigurazione di un piano, con particolare attenzione ai suoi vantaggi rispetto ad una ri-pianificazione da zero.

- Nel capitolo 4 verrà presentato il concetto di Mixed-Initiative Planning, descrivendone le caratteristiche e i principali vantaggi. Verranno inoltre citati alcuni importanti lavori ([2], [3], [4]) al fine di mostrare i benefici dell'approccio misto alla pianificazione. Infine verrà chiarito come il sistema sviluppato in questa tesi adotti tale approccio permettendo una pianificazione interattiva.

- Nel capitolo 5 verrà presentato il sistema sviluppato per questa tesi. In particolare ne sarà data una visione concettuale, soffermandosi sugli obiettivi e sulla descrizione delle principali funzionalità da esso fornite, con particolare attenzione ai loro vantaggi e alla loro importanza.

- Nel capitolo 6 verrà mostrata una visione più concreta del sistema sviluppato. Ne verrà presentata l'architettura implementativa, con riferimento alle principali classi sviluppate e alla loro implementazione. Infine verrà descritta l'interfaccia grafica del sistema in relazione alle sue funzionalità.
- Nel capitolo 7 saranno mostrati tre scenari di utilizzo del sistema al fine di evidenziare ulteriormente i benefici dell'utilizzo di un approccio misto come quello adottato e delle funzionalità di costruzione interattiva di un piano introdotte.

Capitolo 2

Pianificazione con vincoli di risorse

2.1 Pianificazione

A livello pratico, la pianificazione classica si può trattare come un problema di ricerca nello spazio degli stati. Per rappresentare un problema (stato iniziale, goal e operatori) è ormai consuetudine utilizzare linguaggi STRIPS-like.

STRIPS (Stanford Research Institute Problem Solver) è un linguaggio basato sulla logica proposizionale e sull'uso di predicati. Esso permette di rappresentare ogni stato attraverso l'insieme di fluenti (predicati) che sono validi in quel determinato stato. Ad esempio l'espressione

`IN(R1, M5) AND ROAD(M1, M8)`

indica che il rover `R1` si trova in `M5` e che `M8` è raggiungibile da `M1`

Le operazioni (o azioni) sono invece rappresentate attraverso precondizioni ed effetti. Le precondizioni includono i fluenti che devono essere veri affinché si possa applicare l'azione. Gli effetti sono invece le conseguenze dell'applicazione dell'azione, ossia le modifiche che un'azione comporta sul mondo. Un esempio è presentato in figura 2.1.

STRIPS fa un'assunzione di mondo chiuso (*Closed World Assumption*), cioè suppone che ogni fatto non espressamente indicato sia falso. Inoltre, mentre si effettua una operazione, si assume che nel mondo non succeda nient'altro. Per

```
/*azione*/
DRIVE(R, A, B)
/*Precondizioni*/
IN(R, A) AND ROAD(A, B)
/*Effetti*/
IN(R, B) AND NOT(IN(R, A))
```

Figura 2.1: Esempio di precondizioni ed effetti di una azione in PDDL

questa ragione negli effetti di una azione si specificano solamente i cambiamenti che essa comporta, mentre tutto il resto non è esplicitato. Questo è importante in quanto nella maggior parte degli ambienti in cui si utilizza STRIPS, c'è una grande quantità di fatti che non cambiano nel tempo. Specificare solamente ciò che si modifica, dunque, permette di evitare di dover ricopiare completamente l'intero modello del mondo ogni volta che si passa da uno stato ad un altro. Gli effetti di una azione, perciò, sono talvolta indicati anche come due liste: una `DELETE-LIST`, che contiene tutte le clausole che devono essere rimosse (e quindi implicitamente rese false), e una `ADD-LIST`, che contiene tutte le clausole che non sono presenti nello stato corrente e devono essere aggiunte (rese vere). Gli effetti dell'esempio precedente si possono dunque trovare indicati anche come in figura 2.2.

```
/*Effetti*/
 /*Add List*/
 IN(R, B)
 /*Delete List*/
 IN(R, A)
```

Figura 2.2: Add List e Delete List in STRIPS

Una volta definito il problema, il compito del pianificatore è verificare se esiste una sequenza di operatori che faccia passare dallo stato iniziale a quello finale, e in tal caso, dire qual è. Per fare ciò il sistema intelligente esamina lo spazio degli stati, simulando, in ogni stato, l'applicazione di tutte le operazioni applicabili (quelle le cui precondizioni sono soddisfatte), fino ad ottenere lo stato finale. Nel caso riesca a raggiungere il goal, dovrà poi restituire il percorso trovato, ovvero il piano.

2.2 PDDL e fluenti numerici

Intorno agli anni 2000, in seguito alle prime due edizioni dell' International Planning Competition (IPC), si è affermato all'interno della comunità internazionale, come linguaggio standard per la rappresentazione di domini, PDDL, un linguaggio sviluppato da Drew McDermott e da alcuni suoi colleghi [1]. Attraverso tale standardizzazione si è cercato di incentivare il progresso nel campo della pianificazione, favorendo la comunicazione tra i vari gruppi di ricerca, lo scambio e la comparazione dei lavori da essi effettuati.

PDDL (Planning Domain Definition Language), è dunque un Linguaggio di Descrizione dei Domini di Pianificazione. In accordo con quanto espresso in [1], esso si ispira a STRIPS e alla formulazione dei problemi secondo la sua logica. In particolare PDDL contiene STRIPS, e i domini scritti in PDDL si possono scrivere in STRIPS. Ci sono comunque alcune differenze tra i due linguaggi. In PDDL gli effetti di una azione non sono esplicitamente divisi in `ADD-LIST` e `DELETE-LIST` ma gli effetti negativi sono indicati con una negazione e sono messi in congiunzione insieme a quelli positivi. Ad esempio in

```
(and (in r1 l2) (not (in r1 l1)))
```

si può notare l'uso del `not` per indicare che `(in r1 l1)` non è più vera (corrisponde alla `DELETE-LIST`) e che `(in r1 l2)` e `(not (in r1 l1))` siano messi in congiunzione tramite `and`. Tra le principali qualità di PDDL vi è una divisione delle caratteristiche di un dominio da quelle dell'istanza di un problema. Esso permette, cioè, di dividere la descrizione di azioni parametrizzate dalla descrizione di oggetti specifici o di condizioni iniziali e finali. In questo modo è possibile utilizzare delle variabili per parametrizzare le azioni e ciò permette di riutilizzare un dominio con istanze di problemi diversi, favorendo un'analisi di scenari differenti applicati allo stesso dominio.

Un esempio di definizione di dominio in PDDL è riportato in figura 2.3.

In esso si può notare la descrizione (di cui prima) delle azioni parametrizzate (in questo caso una sola: `drive`) al fine di lasciare poi all'istanza del problema il compito di sostituire i parametri formali con dei parametri attuali.

Un' importante caratteristica di PDDL, quindi, riguarda la definizione dei tipi. I tipi dei parametri di una azione vengono specificati esplicitamente all'interno di

`:parameters` in cui una variabile è indicata con il punto interrogativo e il suo tipo è preceduto dal simbolo `-` (e.g. `:parameters ( ?r - robot ?l1 - site ?l2 - site)`). Le stesse variabili definite nell'elenco degli argomenti, sono poi naturalmente utilizzate nella descrizione delle precondizioni e degli effetti dell'azione (e.g `:precondition (and (in ?r ?l1) (road ?l1 ?l2) ... )`).

I nomi dei tipi devono essere dichiarati (prima di essere usati) attraverso `:types` (e.g. `(:types robot - object site - object)`) e se si intende usarli è anche necessario dichiararlo all'interno dei requisiti (e.g. `(:requirements :typing ... )`).

In figura 2.4, invece, viene mostrata la descrizione di un problema molto semplice.

In questo caso, si può notare come le descrizioni degli oggetti specifici (e.g. `m0 -site` che dichiara `m0` come un oggetto di tipo `site`, oppure `r1 -robot` che dichiara `r1` di tipo `robot`), le condizioni iniziali (e.g. `(in r1 m1)`) e i *goals*, vengano indicate senza l'uso di parametri, in quanto stiamo configurando l'istanza di un problema.

I due precedenti esempi sono espressi in una rappresentazione che è già propria di PDDL 2.1 (estensione di PDDL sviluppata in occasione della terza IPC nel 2002, vedi [5]). Vengono già utilizzati, infatti, i fluenti numerici (introdotti proprio con questa versione di PDDL). Elementi di questo tipo sono molto importanti, soprattutto per quanto riguarda la pianificazione di problemi realistici, in quanto permettono di modellare risorse non binarie, come possono essere ad esempio il livello di carburante, l'energia, la distanza o il tempo. Essi sono indicati separatamente dai predicati. All'interno della definizione di un dominio i predicati (e.g `(road ?l -site ?l1 - site)`), infatti, vengono specificati in `:predicates`. Mentre le funzioni numeriche (e.g `(roughness ?l1 -site ?l2 -site)`) sono inserite in `:functions`.

I predicati non presentano dei vincoli numerici. Essi possono essere veri o falsi e il loro significato non è intrinseco, dipende invece dagli effetti che le azioni del dominio possono avere su di essi, o meglio, sullo stato dell'ambiente. Ad esempio, un predicato `in (r1, l1)` può essere vero nello stato `i` ma non nello stato `i+1` ottenuto dall'applicazione di una azione `drive(r1, l1, l2)`. Si possono

distinguere dunque, a livello concettuale (a livello sintattico non ci sono differenze), predicati statici e predicati dinamici. I primi sono predicati come `road l1 l2` (per quanto riguarda il dominio indicato precedentemente), predicati, cioè, che non vengono cambiati da nessuna azione e che quindi dipendono solamente dallo stato iniziale del problema. I secondi invece sono quei predicati il cui valore viene alterato dalle azioni.

Le funzioni, invece, possono essere viste come dei predicati a cui è associato un valore numerico anziché un valore booleano. Anche per esse, in generale, valgono le stesse considerazioni fatte per i predicati.

Sia i predicati che le funzioni possono essere utilizzati come vincoli all'interno delle precondizioni di una azione, e i loro valori possono essere alterati dagli effetti.

I vincoli numerici (e.g `(<= (roughness ?l1 ?l2) 3))`) possono essere costruiti, usando operatori aritmetici, a partire da espressioni numeriche primitive, ossia da valori che sono associati a tuple di oggetti dalle funzioni del dominio. I fluenti numerici permettono di aumentare il potere espressivo di PDDL garantendo la possibilità di una migliore rappresentazione del mondo, attraverso, appunto, valori numerici. In questo modo all'interno di un piano è possibile tener conto di informazioni che sono rappresentate numericamente e che spesso sono fondamentali sia per il corretto funzionamento del sistema sia per la raggiungibilità del goal.

In PDDL 2.1, oltre ai fluenti numerici, è stata anche introdotta la possibilità di definire, attraverso qualsiasi espressione aritmetica, delle metriche. Esse permettono di specificare i criteri su cui un piano deve essere valutato. L'introduzione di tali vincoli è dovuta alla possibilità di utilizzare dei fluenti numerici. Infatti un semplice esempio di definizione di metrica può essere il seguente: `(:metric minimize (powerC))`. Esso permette di chiedere al sistema che il piano generato, oltre ad essere consistente, minimizzi il consumo di energia (`powerC`). Naturalmente l'espressione da minimizzare (o massimizzare) può essere molto complessa, ottenendola dalla combinazione di più funzioni.

2.3 Pianificatori

Come già accennato in precedenza, il compito di un pianificatore è la creazione di un piano di azioni che permetta di raggiungere, a partire dallo stato iniziale

del problema, uno stato in cui le condizioni di goal siano verificate. Per fare ciò il pianificatore, in ogni stato, valuta tutte le azioni presenti all'interno del dominio e simula l'applicazione di quelle applicabili (quelle le cui precondizioni sono soddisfatte), generando un nuovo stato, fino a raggiungere ad uno stato finale.

La pianificazione classica, come visto, non tiene conto dei valori numerici. Un celebre esempio di pianificatore, spesso utilizzato per problemi tradizionali in quanto molto efficiente, è FF.[6] Esso è un sistema realizzato da Joerg, in grado di gestire problemi di pianificazione STRIPS espressi in PDDL. FF è un sofwtare open source, sviluppato in C e pubblicato sotto licenza GNU General Public.

Lo sviluppo di PDDL 2.1, e quindi l'introduzione dei fluenti numerici all'interno di PDDL, ha portato alla necessità di pianificatori che fossero in grado di gestire anche questo tipo di strutture. Un pianificatore di questo tipo, riconosciuto a livello internazionale in quanto si è dimostrato molto efficiente durante la terza International Planning Competition, è Metric-FF.[7] Esso è stato sviluppato da Joerg Hoffmann ed estende FF alle variabili numeriche. Dove FF permette di inserire atomi logici (cioè all'interno delle formule condizionali in precondizioni, effetti e *goals*), Metric-FF permette anche l'inserimento di vincoli numerici, rendendo dunque possibili, ad esempio, confronti numerici (`<, <=, =, >=, >`) tra due espressioni. Allo stesso modo, per quanto riguarda gli effetti di una azione, Metric-FF accetta anche effetti numerici che permettono di aggiornare (`=, +=, -=` , etc.) il valore di una variabile numerica.

All'interno del sistema sviluppato in questa tesi, viene utilizzato, per le funzioni di ricerca di un piano, Metric-FF. I domini su cui si è lavorato, infatti, contengono fluenti numerici, in quanto l'utilizzo delle MMA (vedi capitolo successivo) porta a vantaggi se si ha a che fare con azioni che presentano precondizioni ed effetti numerici.

```
(define (domain space)
  (:requirements :typing :fluents)
  (:types robot  - object site - object)
  (:predicates (in ?r - robot ?l - site)
     (road ?l -site ?l1 - site)
  )
  (:functions (distance ?l1 ?l2 -site)
     (roughness ?l1 ?l2 -site)
     (time)
     (power ?r -robot)
     (powerC ?r -robot)
     (cons)
     (speed)
  )
  (:action drive
     :parameters ( ?r - robot ?l1 - site ?l2 - site)
     :precondition (and (in ?r ?l1) (road ?l1 ?l2)
        (>=(power ?r)
           (/ (* (cons)(/ (* (distance ?l1 ?l2)  100)(speed)))100))
        (<= (roughness ?l1 ?l2) 3))
     :effect (and
        (in ?r ?l2) (not (in ?r ?l1))
        (decrease(power ?r)
           (/ (* (cons)(/ (* (distance ?l1 ?l2) 100)(speed)))100))
        (increase(time)
           (/ (* (distance ?l1 ?l2) 100) (speed)))
        (increase(powerC ?r)
           (/ (* (cons)(/ (* (distance ?l1 ?l2) 100)(speed)))100)))
   )
}
```

Figura 2.3: Esempio di definizione di dominio in PDDL

```
(define (problem problem_S11_G4)
(:domain space)
(:objects m0 -site m1 -site m2 -site m3 -site r1 -robot )
(:init
    (= (distance m0 m1) 18)
    (road m0 m1)
    (= (roughness m0 m1) 1)
    (= (distance m0 m3) 19)
    (road m0 m3)
    (= (roughness m0 m3) 3)
    (= (distance m2 m1) 52)
    (road m2 m1)
    (= (roughness m2 m1) 3)
    (= (distance m2 m3) 39)
    (road m2 m3)
    (= (roughness m2 m3) 2)
    (= (distance m3 m1) 37)
    (road m3 m1)
    (= (roughness m3 m1) 3)
    (= (power r1) 1584.0)
    (= (powerC r1) 0)
    (= (time) 0)
    (= (cons) 50)
    (= (speed) 50)
    (in r1 m1))
(:goal (and (in r1 m3)(< (powerC r1)100.0)(< (time)80.0)))

)
```

Figura 2.4: Esempio di definizione di problema in PDDL

Capitolo 3

MMA - Multi Modality Actions

Nei capitoli precedenti si è parlato di MMA (Multi Modality Actions). Esse sono parte di un concetto sviluppato in [9] come estensione di PDDL.

L'idea nasce principalmente dal fatto che in ambienti reali e non completamente deterministici è facile che un piano, a tempo di esecuzione, venga reso inconsistente a causa di eventi imprevisti. In situazioni di questo tipo l'agente intelligente deve cercare un modo per risolvere il problema.

Una soluzione può essere una nuova pianificazione del problema. Essa però spesso risulta lenta e costosa in termini computazionali, in particolare se deve essere fatta online, ad esempio su un rover dotato di capacità di calcolo limitate. Inoltre una ri-pianificazione del problema può portare ad una modifica del piano originale, mentre spesso è importante che le decisioni e le aspettative di alto livello fornite a tempo di pianificazione (offline) non vengano alterate.

Una alternativa può dunque essere l'utilizzo delle Multi Modality Actions. In alcune situazioni, infatti, è possibile che una azione possa essere eseguita in modi diversi. Le azioni vanno ad impattare l'uso delle risorse del sistema in maniera differente a seconda della modalità di esecuzione. In una situazione in cui il piano correntemente istanziato non sia più valido in quanto le risorse a disposizione non sono più sufficienti per completarne l'esecuzione, si può pensare di cambiare le modalità delle azioni, cercandone una nuova configurazione che richieda una minore o diversa quantità di risorse.

3.1 Le azioni

In figura 3.1 è riportato un esempio di azione multi-modalità tratto dal dominio ZenoTravel (vedi Appendice A.2), il quale è solitamente usato per rappresentare uno scenario di logistica che prevede un insieme di velivoli (`planes` o `aircraft`) con il compito di trasportare delle persone da un luogo ad un altro.

L'azione `fly` permette, all'interno del dominio, lo spostamento di un `aircraft ?a` da una `city ?c1` ad un'altra `city ?c2`. L'azione richiede, per essere eseguita, che l'`aircraft ?a` si trovi nella `city ?c1` (`(located ?a ?c1)`) e una certa quantità di `fuel` (carburante) sull'`aircraft` in relazione alla modalità di esecuzione (vedi `:numPrecondition`). Dopo essere eseguita, viene negato il predicato `(located ?a ?c1)` e viene reso vero il predicato `(located ?a ?c2)` (viene cioè aggiornata la posizione corrente dell'`aircraft`), inoltre vengono aggiornati, a seconda della modalità di esecuzione, i valori indicanti il carburante utilizzato (`total-fuel-used` e `fuel`) e il tempo impiegato (`time-spent`).

Come si può notare dalla figura, alla base delle MMA vi è una netta distinzione tra un livello qualitativo di descrizione delle azioni (al cui interno troviamo precondizioni ed effetti proposizionali) e uno quantitativo (che riguarda requisiti ed effetti numerici). L'azione viene divisa in una parte simbolica, che descrive precondizioni ed effetti proposizionali, i quali sono comuni a tutte le modalità (e.g. `:precondition (located ?a ?c1)` oppure `:effect (and (not (located ?a ?c1)) (located ?a ?c2)))`), e una parte numerica, contenente le precondizioni e gli effetti numerici di ogni modalità, i quali, naturalmente, sono diversi tra loro, altrimenti non vi sarebbe differenza tra le azioni/modalità (ad esempio si noti la differenza tra i requisiti delle due modalità, in `:numPrecondition`).

Le MMA permettono di rappresentare in forma compatta alternative di modalità di esecuzione di una azione. Le modalità di esecuzione dunque rappresentano diversi modi in cui un'azione può essere eseguita. Ognuna di esse raggiunge gli stessi effetti proposizionali mentre si differenziano una dall'altra nelle precondizioni e negli effetti numerici, cioè nel consumo di risorse numeriche.

L'utilizzo di fluenti numerici è dunque di fondamentale importanza. Le MMA infatti sono caratterizzate proprio dai diversi valori di prerequisiti ed effetti numerici. Le novità introdotte in PDDL 2.1 sono quindi indispensabili per l'utilizzo

```
(:action fly
    :parameters (?a - aircraft ?c1 ?c2 - city)
    :modalities {cruise, zoom}
    :precondition (located ?a ?c1)
    :numPrecondition
        (cruise:
            (>= (fuel ?a) (* (distance ?c1 ?c2) (cruise-burn ?a)))
        (zoom:
            (>= (fuel ?a) (* (distance ?c1 ?c2) (zoom-burn ?a))))
    :effect (and (not (located ?a ?c1)) (located ?a ?c2)))
    :numEffect
        (cruise:(and
            (increase (total-fuel-used)
                      (* (distance ?c1 ?c2) (cruise-burn ?a)))
            (decrease (fuel ?a)
                      (* (distance ?c1 ?c2) (cruise-burn ?a)))
            (increase (time-spent)
                      (/ (distance ?c1 ?c2) (cruise-speed ?a)))))
        (zoom:(and
            (increase (total-fuel-used)
                      (* (distance ?c1 ?c2) (zoom-burn ?a)))
            (decrease (fuel ?a)
                      (* (distance ?c1 ?c2) (zoom-burn ?a)))
            (increase (time-spent)
                      (/ (distance ?c1 ?c2) (zoom-speed ?a))))))
```

Figura 3.1: Esempio di azione multi-modalità: fly

delle MMA.

3.2 MMA e PDDL

Le MMA, come detto, sono un'estensione di PDDL. Esse tentano di dare ad esso una strutturazione maggiore, permettendo di scrivere le azioni dividendo le informazioni proposizionali da quelle numeriche.

Talvolta in un dominio PDDL si trovano a coesistere due o più azioni con lo stesso compito, cioè azioni che ottengono gli stessi effetti proposizionali (ad esempio spostano un oggetto `o1` da una posizione `p1` ad una posizione `p2`), ma con requisiti ed effetti numerici differenti. Tali azioni possono essere raggruppate in una unica azione "multi-modalità".

Ad esempio l'azione `fly` della precedente figura, è stata ricavata da due azioni (`fly` e `zoom`) aventi gli stessi effetti e le stesse precondizioni proposizionali. È stato dunque possibile unirle in una unica azione con due diverse modalità di esecuzione (`cruise` e `zoom`).

Poiché viene mantenuta la compatibilità con PDDL è possibile convertire una MMA in un insieme di azioni PDDL e, viceversa, un insieme di azioni PDDL può essere "compattato" (se precondizioni ed effetti proposizionali di ogni azione coincidono) in una unica MMA.

Questo è importante in quanto permette di scrivere ed utilizzare dei piani contenenti MMA (anche chiamati MMP - Multi Modality Plans) con sistemi che sfruttano PDDL. Una azione multi-modalità può infatti essere scomposta in un insieme di azioni PDDL, ognuna delle quali determinerà il comportamento del sistema quando la corrispondente modalità è selezionata. Analogamente, a partire da un insieme di azioni PDDL aventi tutte le stesse caratteristiche simboliche, è sufficiente creare un'unica MMA che contenga tante modalità quante sono le azioni PDDL selezionate, con le loro relative caratteristiche numeriche.

Un esempio può essere tratto dal dominio dei rover (vedi Appendice A.1). Tra le altre, abbiamo due azioni (riportate in figura 3.2) che permettono al rover di scattare una fotografia rispettivamente a bassa e ad alta risoluzione.

Utilizzare l'azione `tp-hr`, piuttosto che `tp-lr`, permette di scattare una foto ad una risoluzione maggiore (quindi con una minore perdita di informazioni), compor-

```
(:action tp-lr
     :parameters ( ?r - robot ?l1 - site )
     :precondition (and(in ?r ?l1) (>= (memory ?r) 8))
     :effect
      (and
      (info ?r ?l1)
      (increase (memoryC ?r) 8)
      (decrease (memory ?r) 8)
      (increase (time) 1)
      (increase (infoLoss) 3)))

(:action tp-hr
     :parameters ( ?r - robot ?l1 - site )
     :precondition (and(in ?r ?l1) (>= (memory ?r) 16))
     :effect
      (and
      (info ?r ?l1)
      (increase (memoryC ?r) 16)
      (decrease (memory ?r) 16)
      (increase (time) 2)
      (increase (infoLoss) 0)))
```

Figura 3.2: Azioni tp-lr e tp-hr del dominio dei rover

```
(:action tp
    :parameters ( ?r - robot ?l1 - site )
    :modalities (lr,hr)
    :precondition (and(in ?r ?l1)
        (lr: (>= (memory ?r) 1))
        (hr: (>= (memory ?r) 2)))
    :effect (and (info ?r ?l1)
        (lr:
            (increase (memoryC ?r) 1)
            (decrease (memory ?r) 1)
            (increase (time) 1)
            (increase (infoLoss) 3))
        (hr:
            (increase (memoryC ?r) 2)
            (decrease (memory ?r) 2)
            (increase (time) 1)
            (increase (infoLoss) 1)))
)
```

Figura 3.3: Azione multi-modalità tp

tando però un maggior consumo di memoria e richiedendo un tempo di esecuzione superiore. Per queste ragioni le due azioni hanno dei prerequisiti e degli effetti numerici diversi, per esempio `tp-hr` necessita di un numero di slot di memoria superiore a 16 (`(>= (memory ?r) 16)`), mentre per `tp-lr` ne sono sufficienti 8 (`((>= (memory ?r) 8))`). Tuttavia notiamo che entrambe le azioni hanno, da un punto di vista proposizionale, le stesse precondizioni (`(in ?r ?l1)`) e gli stessi effetti (`(info ?r ?l1)`). Possiamo dunque raggruppare queste due azioni PDDL in una unica MMA, come mostrato in figura 3.3. Notiamo che le modalità dell'azione sono elencate in `:modalities` e che le precondizioni e gli effetti contengono una parte iniziale comune a tutte le modalità (la parte proposizionale) e una parte numerica che viene suddivisa in modalità, e per ognuna di esse vengono specificati gli opportuni valori.

3.3 Riconfigurazione

La possibilità di avere più modalità di esecuzione di una azione (come ad esempio due differenti dispositivi per effettuare una comunicazione) garantisce all'agente una maggiore versatilità nell'esecuzione di un piano e nell'affrontare eventuali imprevisti. Il risultato richiesto è infatti ottenibile in modi diversi, in base alla configurazione di una azione.

In particolare, durante l'esecuzione di un piano in un ambiente reale, la variazione del valore di risorse numeriche può essere soggetta ad imprevisti. In questi casi l'esecuzione di una azione con una determinata modalità può non essere possibile a causa della scarsa disponibilità di un certo tipo di risorsa. La stessa azione, però, potrebbe invece essere eseguita con un'altra modalità che necessita di una quantità minore delle risorse attualmente "critiche", e che richieda una quantità maggiore di un tipo di risorse che sono invece disponibili in quantità più che sufficiente.

In caso di contingenze a tempo di esecuzione che rendano il piano numericamente inconsistente, è dunque possibile fare un primo tentativo di riconfigurazione delle modalità di esecuzione delle azioni che devono ancora essere eseguite.

La riconfigurazione di un piano online parte dallo stato corrente del mondo e dell'agente. Tutte le azioni già effettuate non vengono più considerate. Si valutano invece i valori delle risorse correnti, gli obiettivi da raggiungere e le azioni rimanenti nel piano già costruito. L'operazione di riconfigurazione effettua una ricerca di una soluzione che abbia la stessa sequenza ordinata di azioni di quella del piano originale, che però presenti una diversa istanziazione delle modalità che soddisfi tutti i vincoli numerici. Viene dunque cercata una nuova configurazione delle modalità di esecuzione delle azioni rimanenti da eseguire, tale che rispetti i vincoli numerici, a differenza di quella attuale resa ormai inconsistente.

Tale riconfigurazione, come già accennato, è meno costosa di una ri-pianificazione da zero. La ricerca di una soluzione, in questo caso, si limita infatti alle modalità delle azioni, sfruttando la struttura generale del piano già costruita. In [10], inoltre, viene mostrato come una soluzione di questo tipo oltre a richiedere, rispetto ad una ri-pianificazione da zero, minor tempo e un uso più contenuto delle risorse computazionali dell'agente, permetta di mantenere la struttura ad alto livello del piano, rispettando l'ordine delle azioni imposto a tempo di pianificazione e

favorendo ad esempio il raggiungimento, da parte di eventuali altri agenti presenti nell'ambiente, di obiettivi dipendenti dall'esecuzione di determinate azioni. Inoltre spesso le conseguenze delle azioni non sono completamente chiare e i comportamenti dell'agente possono diventare difficili da predire, pertanto può capitare di trovarsi in situazioni inaspettate dopo la creazione di un nuovo piano.

Al fine di favorire lo sviluppo di piani che utilizzino MMA, il sistema realizzato in questa tesi è stato implementato in modo da permettere la gestione delle modalità di esecuzione. È infatti possibile, durante la costruzione del piano, decidere quali modalità assegnare alle azioni o modificare quelle già stabilite.

All'interno del sistema il concetto di riconfigurazione viene utilizzato in maniera più estesa rispetto al suo contesto nativo (ovvero online in caso di impossibilità di proseguire l'esecuzione del piano corrente). Come verrà meglio dettagliato nella sezione 5.2 l'operazione viene infatti applicata all'intero piano, ed è in grado di generare una nuova configurazione delle modalità di esecuzione per tutte le azioni. Questo approccio permette di effettuare dei test di consistenza numerica del piano, eventualmente valutando più varianti di configurazione di modalità al fine di decidere quale grado di sicurezza lasciare all'agente durante l'esecuzione.

Inoltre tale concetto viene ulteriormente adattato ad un contesto mixed-initiative (vedi capitolo 4) permettendo il suo utilizzo solo su determinate azioni decise dall'utente. Ovvero l'utente ha anche la possibilità di selezionare alcune azioni per le quali richiedere una configurazione automatica delle modalità da parte del sistema, fissandone altre.

Capitolo 4

Pianificazione interattiva

Durante la creazione di un piano è piuttosto comune che non si conoscano alla perfezione sin dall'inizio informazioni come i vincoli di stato iniziale e stato finale, la sequenza ottimale di azioni, le eventuali situazioni da affrontare, e così via. Un approccio classico alla pianificazione, in casi di questo tipo, non è particolarmente di aiuto. Classicamente la pianificazione è stata infatti affrontata con un approccio o completamente manuale oppure completamente automatico. Ciò è possibile quando il mondo in cui si va ad operare è perfettamente definito e non è particolarmente complesso (per una pianificazione manuale) oppure si è sicuri che un piano sviluppato da un sistema in modo automatico rispecchi alla perfezione le aspettative umane (per una pianificazione automatica).

Una pianificazione completamente manuale, infatti, delega all'uomo l'intera gestione dei problemi con il rischio che l'operatore umano non sia in grado di affrontarli o di trovare una soluzione ottima. Essa è preferibile solamente per problemi riguardanti domini completamente definiti e deterministici e non particolarmente complessi, per i quali l'uomo è in grado di avere una chiara visione degli obiettivi e di tutte le componenti dell'ambiente.

Una pianificazione completamente automatica, d'altro canto, pur permettendo la risoluzione di problemi complessi con grandi quantità di elementi e variabili, è, come quella manuale, vincolata alle conoscenze fornite in input al sistema intelligente. Se tali informazioni non possono essere complete, la pianificazione non è possibile. Se il dominio in cui si lavora non è completamente deterministico,

inoltre, anche in questo caso il piano non potrà possedere una robustezza tale da renderlo versatile ad eventuali contingenze durante l'esecuzione. Inoltre un approccio di questo tipo non permette all'uomo di intervenire su un piano creato dal sistema e dunque è limitato dalla conoscenza e dall'intelligenza del sistema stesso.

Uno dei principali problemi della pianificazione classica risulta quindi essere la mancanza di informazioni sicure e abbastanza dettagliate sul mondo. C'è bisogno di un approccio diverso, che permetta di delegare al sistema intelligente operazioni difficili e dispendiose per l'uomo (verificare l'ottimalità e il soddisfacimento di tutti i vincoli del piano e l'utilizzo delle risorse, cercare un piano per un determinato problema che eventualmente ottimizzi una certa funzione obiettivo, etc.) e che allo stesso tempo lasci all'operatore umano un certo margine di controllo che gli permetta di prendere delle decisioni ed effettuare modifiche in base a quanto emerso dalle operazioni già effettuate o da informazioni aggiuntive sul mondo.

Una pianificazione interattiva, come approccio mixed-initiative alla pianificazione, permette all'operatore di costruire un piano "insieme" al sistema. L'uomo può decidere di scrivere un piano (o parte di esso) manualmente e chiedere al sistema di verificarne la consistenza o di ottimizzarlo secondo una determinata funzione obiettivo oppure può chiedere semplicemente al sistema di effettuare esso stesso la pianificazione, per poi andare ad analizzare il piano ottenuto. L'utente è inoltre in grado di modificare un piano già realizzato cambiando i vincoli, le azioni e il loro ordine, oppure cambiando i *goals* del problema per controllare la validità del piano in condizioni diverse da quelle iniziali. Un approccio di questo tipo prevede dunque un'alternanza di azioni tra l'operatore umano e il sistema intelligente, cioè una interazione tra di essi.

Come ormai dimostrato in vari studi (si vedano ad esempio [2] e [3]), un approccio mixed-initiative alla pianificazione porta a benefici tangibili sulla qualità di un piano. Inoltre permette all'utente di avere un maggiore controllo sul piano, sfruttando le comodità offerte dalla potenza computazionale dei calcolatori. Questa combinazione di funzionalità di *reasoning* automatiche con conoscenze umane non esprimibili all'interno della rappresentazione di un dominio, portano alla costruzione di piani di alta qualità garantendo comunque all'utente un adeguato potere decisionale su quanto viene prodotto.

Una pianificazione interattiva permette dunque di realizzare un piano il più possibile robusto e sicuro che tenga conto di tutte le possibili situazioni prevedibili che potrebbero verificarsi. Naturalmente a tempo di esecuzione, in ambienti non deterministici e non completamente conosciuti, possono verificarsi eventi imprevedibili durante la pianificazione. Rispetto ad una pianificazione classica però, la pianificazione interattiva permette di costruire piani più elastici grazie alla maggiore consapevolezza sul mondo dell'uomo. Durante l'esecuzione verrà applicato il piano finché possibile, e poi, se necessario, verrà modificato per essere adattato alla situazione corrente. Più il piano è elastico più potrà essere modificato senza perdere gli obiettivi di alto livello, garantendo il raggiungimento di tutti i goal richiesti.

Inoltre, più un piano è elastico, più evita che l'agente intelligente debba effettuare online delle operazioni di ricerca di una nuova soluzione del problema all'interno dello spazio degli stati. Compiti di questo tipo infatti, come già detto, sono molto costosi in termini computazionali e perciò possono determinare la riuscita (o la non riuscita) di una missione. Le cosiddette operazioni di *search*, infatti consistono in una ricerca di una sequenza di azioni che, attraverso i loro effetti e rispettando le loro precondizioni, permettano di raggiungere tutti gli obiettivi richiesti. Una ricerca di questo tipo, a seconda del problema da affrontare, può risultare estremamente costosa, richiedendo quantità di tempo talvolta impraticabili, poiché lo spazio degli stati da analizzare può essere enorme. Quando tali operazioni sono però indispensabili, un buon piano dovrebbe richiedere la minima quantità di risorse per essere modificato in una versione diversa ma ancora praticabile. Per questa ragione la pianificazione interattiva è molto utile offline, infatti essa permette di utilizzare la maggior parte delle risorse computazionali in un momento in cui, normalmente, non ci sono particolari problemi di capacità di calcolo o di tempo, realizzando un piano il più versatile possibile grazie alla possibile analisi di vari scenari di esecuzione e dell'uso delle risorse.

Un chiaro esempio di quanto appena detto lo si ritrova nel sistema MAPGEN [2], sviluppato per la missione di esplorazione di Marte iniziata nel 2004 come componente per le operazioni di terra. Esso è stato usatonelle prime fasi della missione per la pianificazione giornaliera delle attività del rover spedito su Marte, attraverso

un approccio mixed-initiative. MAPGEN ha permesso la costruzione dei piani di esecuzione con un procedimento interattivo tra l'utente e il sistema. La pianificazione avveniva in maniera incrementale, alternando operazioni automatiche di generazione di un piano (a partire da quello sviluppato fino a quel momento) a fasi di modifica del piano costruito. MAPGEN fornisce servizi di pianificazione che danno all'utente diversi gradi di controllo. L'operatore può infatti agire delegando al sistema l'intera pianificazione (operazione chiamata *plan-all*) oppure richiedere l'inserimento di un insieme di operazioni in qualunque punto (*plan-selected*) o in una posizione precisa (*place-selected*) del piano corrente, oppure ancora può spostare una azione all'interno di un possibile intervallo temporale (calcolato dal sistema) che mantiene la validità del piano corrente (*constrained-move*) o in una posizione qualsiasi (*super-move*). MAPGEN comunque è stato sviluppato in modo da mantenere al suo interno una struttura sempre aggiornata che non può essere resa inconsistente, pertanto alcune operazioni (e.g. la *super-move*) vengono effettuate solamente se rispettano tutti i vincoli interni, altrimenti vengono annullate. Altre invece vengono tenute in considerazione come euristiche. Ad esempio il sistema considera la posizione di una azione fornita con una *place-selected* come un'euristica durante la ricerca di una soluzione per il problema.

Un sistema come MAPGEN permette dunque all'utente di essere responsabile delle decisioni di alto livello (come la scelta dell'insieme di azioni da inserire nel piano), delegando al sistema le operazioni complesse ed eventualmente influenzandone le scelte. Allo stesso tempo viene alleggerito il carico di vincoli e informazioni che l'utente deve fornire al sistema, come ad esempio il fatto che prima di dover utilizzare un dispositivo elettronico sia necessario accenderlo, e dopo averlo usato spegnerlo. Azioni di questo tipo vengono infatti calcolate dal sistema e automaticamente inserite all'interno del piano.

In questo modo all'utente viene presentato il problema ad un livello più alto e più facilmente comprensibile, allontanandolo dall'implementazione vera e propria pur lasciandogli un elevato grado di controllo.

Un approccio di questo tipo, che permette all'utente di agire ad un livello di astrazione maggiore rendendo la pianificazione più intuitiva, porta, come viene anche presentato in [3] e [4], ad una maggiore efficienza. Un piano costruito attraverso un procedimento mixed-initiative è mediamente migliore di uno sviluppato

in modo completamente automatico dal sistema, per via della conoscenza e della consapevolezza umana che non è possibile implementare all'interno di un software. Se oltre a ciò il sistema fornisce un'interfaccia utente intuitiva attraverso una metafora di pianificazione più facile da comprendere, le prestazioni e la qualità del piano aumentano ulteriormente.

Il sistema GTrans sviluppato da Michael T. Cox e dal suo *team* [3], è una interfaccia mixed-initiative per la realizzazione di piani attraverso la manipolazione dei *goals*. Esso mostra come un approccio che nasconda all'utente gli algoritmi di pianificazione e le rappresentazioni interne, mostrando solamente gli elementi di alto livello rappresentanti gli obiettivi da raggiungere, possa migliorare la qualità di un piano e l'efficienza durante la pianificazione. Il sistema in questione si basa sull'idea che l'utente possa costruire un piano manipolando principalmente i *goals*, senza doversi preoccupare di come venga rappresentato il problema internamente o di quali siano i criteri di ricerca delle soluzioni, e allo stesso tempo permette di ottenere prestazioni maggiori in termini di pianificazione. L'utente ha infatti una visione più chiara del mondo se utilizza un'interfaccia visivamente più intuitiva ed è in grado di valutare lo stato del problema da un punto di vista più astratto, con la possibilità di raggiungere considerazioni che un calcolatore non è in grado di fare.

Il sistema sviluppato in questa tesi, come verrà dettagliato nel capitolo successivo, sviluppa il concetto appena presentato dell'approccio mixed-initiative alla pianificazione, permettendo all'utente di realizzare attraverso una interfaccia grafica *user-friendly* un piano, avvalendosi del supporto di funzionalità automatiche di *reasoning* e avendo la possibilità di modificare il piano in ogni sua componente. L'operatore sarà dunque in grado di effettuare le modifiche che più ritiene opportune, a partire dal numero e dal tipo di azioni da effettuare fino alla scelta delle modalità di esecuzione o dei parametri di una azione. Il sistema rende possibile quanto detto fino ad ora sulla pianificazione interattiva, e da ciò ne conseguono i vantaggi sopra presentati, tra cui l'importante possibilità di testare varianti di un piano su di un problema e/o varianti di un problema su un piano in maniera agevole. Funzionalità che fornisce una migliore visione globale dell'ambiente grazie ad una analisi facilitata di più scenari.

Inoltre, tenendo conto del fatto che a tempo di esecuzione un piano possa es-

sere reso inconsistente da eventuali consumi di risorse imprevedibilmente troppo elevati, il sistema sviluppato, per ridurre la necessità di una ri-pianificazione online, permette durante la pianificazione la creazione di un piano che utilizzi azioni multi-modalità. A tal scopo le MMA risultano essere una risorsa particolarmente preziosa. Configurando le modalità delle azioni in modo da lasciare il più ampio margine di cambiamenti possibili, si crea un piano in grado di risolvere il problema richiesto e che, in caso di imprevisti, permette almeno una riconfigurazione delle modalità che allontana la necessità di dover riformulare l'intero piano.

Dal punto di vista della pianificazione interattiva il sistema mette a disposizione un'operazione di configurazione delle modalità del piano costruito (vedi 5.2). Tale operazione permette all'utente di iniziare a configurare un piano solamente con alcune modalità per poi richiedere al sistema di completare, se possibile, la configurazione di quelle mancanti. Una funzionalità di questo tipo è alla base dell'approccio mixed-initiative in cui vi è un'interazione continua tra l'utente e il calcolatore e permette all'utente di personalizzare il più possibile il piano, delegando al sistema le operazioni più complesse (trovare le modalità applicabili per le azioni per cui non sono state specificate, rispettando tutti i vincoli numerici del problema).

Capitolo 5

Il sistema sviluppato

In questa tesi è stato sviluppato un sistema grafico in grado di dare supporto all'operatore durante la fase di pianificazione. L'obiettivo del sistema è permettere un approccio misto nella realizzazione di un piano, ossia permettere una pianificazione interattiva.

Il software sviluppato fornisce infatti al pianificatore strumenti che gli permettono di capire facilmente in che condizioni si trova il piano e di modificarlo attraverso funzioni di controllo, alterazione ed esecuzione di ciò che è stato realizzato. L'operatore ha la possibilità di costruire "graficamente" un piano da zero, scegliendo quali azioni inserire, modificare o eliminare, e quali modalità di esecuzione associare ad esse. Può inoltre, tra le altre possibilità, chiedere al sistema di verificare la validità del piano costruito (dal punto di vista simbolico e/o numerico) o di richiederne una riconfigurazione delle modalità.

Il progetto è una estensione del lavoro sulle MMA del Dott. Scala [9], e, come mostrato in figura 6.1, ne richiama ed integra gli strumenti realizzati, per permettere all'utente una visualizzazione ed una elaborazione delle informazioni necessarie in modo semplificato e *user-friendly*. Il sistema, per tale ragione, sfrutta le librerie *MMACSP* e *PPMajal* e ne arricchisce le funzionalità per adattarle ad una pianificazione offline interattiva. Questo consente la realizzazione di piani che durante l'esecuzione, in caso di necessità, possono essere riconfigurati, con i relativi vantaggi visti nel capitolo 3.

Il sistema è indipendente dal dominio su cui si vuole lavorare, pertanto è pos-

sibile operare su qualsiasi dominio, problema o piano purché essi siano espressi in PDDL. Esso inoltre è in grado di fornire, al termine della pianificazione, piani eseguibili a tutti gli effetti da agenti intelligenti.

Il capitolo corrente ha lo scopo di presentare il progetto e fornisce una visione concettuale del sistema esponendone gli obiettivi e le principali funzionalità. Nel capitolo 6 verrà fornita, invece, una visione più tecnica concentrandosi maggiormente sui dettagli implementativi.

5.1 Obiettivi

L'obiettivo principale che si è cercato di raggiungere durante lo sviluppo è permettere un approccio mixed-initiative alla pianificazione. Come espresso nel capitolo precedente, esso consente all'operatore di costruire manualmente un piano e di effettuare su di esso le modifiche che ritiene opportune, oppure allo stesso tempo di richiamare funzioni automatiche del sistema che vanno dal controllo di esistenza di un piano alla richiesta di ottimizzazione secondo un certo criterio. Un approccio misto di questo tipo, come già detto, garantisce all'utente (l'operatore) la possibilità di una pianificazione facilitata grazie alle funzioni automatiche in grado di semplificare il lavoro risolvendo in tempi brevi compiti che manualmente richiederebbero molto tempo. Inoltre permette all'operatore di analizzare i dati del piano costruito fino ad un determinato momento, confrontarli con quelli di altri piani o di altri problemi e decidere come modificare il piano per migliorarlo o completarlo.

Un'interfaccia *user-friendly* supporta l'utente fornendo una visione facilitata di quale sia la situazione corrente di un piano e dà la possibilità di strutturarlo utilizzando direttamente le componenti del sistema, senza dover scrivere manualmente le azioni. Questo, oltre a rendere il processo più immediato e meno problematico, semplifica il lavoro, evitando all'operatore di doversi ricordare perfettamente la sintassi PDDL, riducendo il rischio di errori non concettuali.

5.2 Funzionalità

Le funzionalità fornite dal sistema sono molteplici. Di seguito sono presentate quelle principali.

1. **Costruzione manuale di un piano**

   La funzionalità di base del sistema è la costruzione manuale di un piano. In questo contesto il termine "manuale" non va inteso nella sua accezione di "a mano" bensì come identificazione di un processo di costruzione non automatico, cioè realizzato passo-passo dall'operatore.

   La creazione manuale di un piano, come funzionalità di alto livello, racchiude al suo interno gran parte delle operazioni effettuabili su di un piano. Esse verranno dettagliate maggiormente nella descrizione delle successive funzionalità.

   L'operazione prevede per l'utente la possibilità di aggiungere una azione alla volta ad un piano inizialmente vuoto, a partire da quelle definite all'interno del dominio. L'operatore può istanziare determinati oggetti del problema come parametri attuali dello schema dell'azione scelto, può decidere la posizione di inserimento dell'azione all'interno del piano e può specificare o meno la sua modalità di esecuzione.

   Ogni componente dell'azione (modalità e parametri) potrà naturalmente essere modificato successivamente, così come la posizione dell'azione stessa all'interno del piano (ovvero la sua posizione nella sequenza ordinata di azioni che formano il piano) o addirittura la sua esistenza (ossia sarà possibile decidere se rimuoverla dal piano). Nel caso non si specifichi la modalità di esecuzione, inoltre, sarà possibile in seguito chiedere al sistema di effettuare una configurazione delle modalità non impostate.

   Al termine della (o durante la) costruzione del piano sarà poi possibile effettuare operazioni di check per verificare la consistenza (proposizionale e/o numerica) di quanto realizzato.

   Quando l'utente lo riterrà opportuno potrà infine salvare su file il piano costruito fino a quel momento, per poterlo utilizzare in altri ambienti o per farlo eseguire da un agente intelligente.

2. **Modifica di un piano**

La funzionalità di modifica di un piano consiste nella possibilità di effettuare varie operazioni su di stesso. Ci sono quattro principali operazioni di modifica:

(a) **Inserimento di una azione** L'inserimento di una azione, come anticipato sopra, permette all'utente di aggiungere al piano una nuova azione.

All'operatore viene chiesto di scegliere quale azione inserire tra tutte quelle disponibili nel dominio. Effettuata la scelta, il sistema richiede la selezione dei valori da assegnare ai parametri (permettendo naturalmente la scelta solo tra gli oggetti presenti nel problema che hanno un tipo compatibile con quello del parametro) se necessari, della posizione di inserimento dell'azione all'interno del piano e di una modalità di esecuzione (non obbligatoria).

Dopo aver completato tale operazione il piano verrà aggiornato con i dati della nuova azione e sarà possibile proseguire nella pianificazione.

Occorre notare che un'azione inserita in questo modo è corretta dal punto di vista formale, tuttavia può contenere degli errori sia in relazione alle altre azioni del piano, sia come azione a se stante. Una azione può infatti non essere valida se le sue precondizioni non sono verificate nello stato i del piano.

Una azione `drive(r1, l1, l2)` inserita in posizione i nel piano non è valida proposizionalmente se nello stato i il `robot r1` non si trova in `l1` o se non esiste una strada tra `l1` e `l2` e se tutte le precondizioni numeriche dell'azione `drive` nello stato i non sono verificate.

Pertanto la posizione di inserimento dell'azione è determinante per la sua correttezza(validità) all'interno del piano e determina anche la correttezza delle azioni successive.

(b) **Rimozione di una azione** La rimozione di una azione permette all'utente di cancellare dal piano una azione precedentemente inserita.

La cancellazione comporta, oltre alla rimozione dal piano delle informazioni riguardanti l'azione, lo slittamento all'indietro di tutte le azioni successive.

Naturalmente, anche per la rimozione di una azione valgono le stesse considerazioni fatte per l'inserimento, ovvero: se prima della cancellazione di una azione il piano costruito era corretto dal punto di vista simbolico e numerico (cioè valido), dopo la rimozione ciò potrebbe non essere più vero in quanto le precondizioni delle azioni successive a quella eliminata potrebbero dipendere dagli effetti di quest'ultima.

E' necessario dunque prestare attenzione all'operazione di eliminazione di una azione poiché le sue conseguenze non sono banali e neppure chiaramente visibili, si possono infatti ripercuotere anche su azioni a lunga distanza da quella rimossa rendendo il piano "inaspettatamente" non più valido.

(c) **Scambio di due azioni** Lo scambio di due azioni permette all'utente di effettuare un'operazione di *swap*, cioè di scambiare la posizione di due azioni all'interno del piano.

Uno scambio non comporta slittamenti di alcun tipo nel piano, semplicemente al posto dell'azione `i` viene inserita l'azione `j` e viceversa.

Ciò non previene però il rischio di inconsistenze numeriche o proposizionali, per le stesse ragioni descritte per la rimozione di una azione. Anche questa operazione, perciò, è particolarmente delicata ed è necessario utilizzarla consapevolmente. Infatti anche due azioni "vicine" nel piano, cioè in posizioni `i` e `i+1`, se scambiate, possono portare ad effetti completamente diversi se dipendono una dall'altra.

Supponendo di avere due azioni `A`: `drive(r1 l1 l2)` e `B`: `drive (r1 l2 l3)` e che il `rover r1` si trovi in posizione `l1`, se inserite nell'ordine `A-B`, le azioni hanno come effetto proposizionale globale lo spostamento di `r1` da `l1` a `l3` e come effetti numerici i relativi consumi di carburante e tempo. Se esse vengono scambiate, invece, nessuna delle due potrà essere eseguita. `B`, infatti richiede che `r1` si trovi in `l2`, ma esso è in `l1`. Dunque il piano risulterà non valido a livello proposizionale e non

si eseguirà neppure l'azione `A` e tutte quelle successive.

Naturalmente l'inapplicabilità di una azione può presentarsi anche a distanza di più azioni, anche per cause non esplicite come in questo caso, come ad esempio un certo utilizzo di risorse al posto di un altro, che porta ad effetti collaterali dopo un indeterminato numero di azioni.

(d) **Modifica di una azione** La modifica di una azione permette all'utente di cambiare la modalità e i parametri ad essa associati.

L'operatore ha la possibilità di modificare la modalità di esecuzione, scegliendo tra quelle specificate nella definizione dell'azione all'interno del dominio, e i parametri, cioè gli oggetti richiesti come argomenti, a partire da tutti quelli presenti nel problema con il tipo richiesto.

La modifica dei parametri comporta le stesse conseguenze descritte per le due operazioni precedenti, determinando la validità proposizionale e/o numerica del piano. La modifica della modalità di esecuzione, invece, ha a che fare, come descritto nel capitolo 3, solamente con precondizioni ed effetti numerici di una azione. Pertanto determina la validità del piano esclusivamente a livello numerico.

Cambiare la modalità di una azione fa sì che essa venga eseguita in un modo differente (mantenendo però stessi requisiti e conseguenze simbolici). Un'operazione di questo tipo ha dunque un impatto meno forte sul piano rispetto ad una modifica dei parametri o alla rimozione, allo scambio e all'inserimento di azioni. Allo stesso tempo, però, modificare anche solo la modalità di esecuzione di una azione a inizio piano potrebbe rendere un piano numericamente valido o migliorarne (anche significativamente) le prestazioni in termini di consumo di risorse, oppure, in generale, potrebbe garantire maggiore versatilità del piano a tempo di esecuzione.

La modifica di una modalità, comunque, può avere anche conseguenze negative sul piano. Un'azione di `drive` con modalità `safe`, ad esempio, permette ad un `rover` di consumare una minore quantità di carburante rispetto ad una esecuzione con modalità `speed`, tuttavia richiede un tempo maggiore. Pertanto il piano ottenuto potrebbe rispettare i vincoli

numerici riguardanti il consumo di carburante, ma non quelli relativi ad eventuali vincoli di tempo.

Anche la modifica di una modalità di esecuzione, dunque, è un'operazione le cui conseguenze non sempre sono immediate o visibili ed è bene tenere conto di queste considerazioni durante la modifica del piano.

3. **Generazione automatica di un piano**

La funzionalità di generazione automatica di un piano permette all'operatore di delegare al sistema la ricerca di un piano che risolva il problema su cui si sta operando.

Il sistema sviluppato utilizza il pianificatore metricFFForMMA (un'estensione di Metric-FF il quale, come già mostrato, permette la gestione di domini e problemi espressi in PDDL che contengono anche fluenti numerici). Tale *planner*, sviluppato da Scala in [9] e fornito all'interno della libreria MMACSP, permette l'utilizzo e la gestione delle azioni multi-modalità.

Il pianificatore elabora, se possibile, un piano, eventualmente ottimizzato secondo una metrica specificata nel problema. Se esiste una soluzione essa verrà restituita all'operatore. Il piano ottenuto sarà valido sia simbolicamente sia numericamente, pertanto potrà essere direttamente salvato su file per essere poi eseguito, senza la necessità di ulteriori operazioni. All'utente verrà comunque data la possibilità di modificarlo come meglio ritiene opportuno attraverso le operazioni di modifica viste sopra.

Poichè il *planner* supporta l'utilizzo delle MMA, il piano costruito presenterà anche un'istanziazione delle modalità. L'operatore potrà modificarla adattandola al meglio ad eventuali esigenze non facilmente specificabili attraverso i vincoli del goal, oppure per confrontarla con altre istanziazioni per valutare i diversi consumi di risorse.

Questa funzionalità permette dunque all'utente di non doversi preoccupare di costruire un piano che sia valido e che soddisfi i goal, l'operatore può così limitarsi ad alterare quanto già elaborato dal sistema.

4. **Check**

Questa funzionalità permette di verificare la consistenza del piano costruito.

Supponendo di avere un piano composto da n azioni $A_0 ... A_{n-1}$, uno stato iniziale del problema S_0 e un insieme di goal G, gli effetti di una azione A_i, se essa è applicabile nello stato S_i (ovvero se le sue precondizioni sono verificate in tale stato) porteranno ad uno stato S_{i+1}.

Un piano è consistente se tutte le precondizioni di ogni azione A_i sono verificate nello stato S_i e se lo stato finale S_{n+1} (ottenuto dall'applicazione in sequenza di tutte le azioni del piano) è uno stato goal, ovvero rispetta tutte le condizioni di G.

Verificare la consistenza di un piano consiste dunque nel simulare l'esecuzione del piano stesso controllando che tutte le sue azioni siano applicabili e che lo stato finale soddisfi i goal.

L'applicabilità di una azione dipende sia dalle sue precondizioni simboliche, sia da quelle numeriche.

All'interno del sistema è possibile richiedere un *check* completo del piano, che prevede un primo controllo di consistenza proposizionale a seguito del quale viene valutata la necessità di controllare anche la consistenza numerica o meno. Se un piano non è valido simbolicamente, infatti, un controllo di validità numerica perde di significato. Viceversa un piano simbolicamente valido può esserlo o meno anche numericamente.

È comunque anche possibile effettuare un controllo di consistenza esclusivamente a livello proposizionale o numerico. Naturalmente non è possibile effettuare un *check* numerico se il piano non è simbolicamente valido.

Il *check* numerico, così come descritto fino ad ora, è possibile solamente per un piano completamente istanziato. Se invece il piano corrente è un piano parzialmente istanziato (in cui non tutte le modalità di esecuzione delle azioni sono state definite) viene messa a disposizione una funzionalità di controllo della configurabilità delle azioni mancanti. Ovvero viene fornita la possibilità di verificare se esiste una possibile configurazione delle modalità che non sono state specificate nel piano. Ovviamente anche questo controllo, così come quello numerico, è possibile solamente nel caso in cui il piano sia valido a livello proposizionale.

(a) **Check proposizionale** Il *check* proposizionale, come già anticipato, verifica che il piano sia valido a livello simbolico. Viene cioè effettuato un controllo di tutte le precondizioni simboliche delle azioni inserite nel piano.

Per ogni azione si analizzano le precondizioni verificando se sono verificate nello stato del problema in cui l'azione deve essere applicata. In tal caso si prosegue l'analisi simulando l'applicazione degli effetti simbolici dell'azione allo stato del problema, fino ad arrivare allo stato finale. A tal punto si verifica che anche le condizioni simboliche del goal siano verificate.

Il piano sarà valido proposizionalmente se tutte le precondizioni e i goal simbolici sono verificati.

(b) **Check numerico** Il *check* numerico verifica che il piano sia valido a livello numerico. Viene cioè effettuato un controllo di tutte le precondizioni numeriche delle azioni inserite nel piano.

Questo test viene effettuato solamente se il piano è valido a livello proposizionale, in caso contrario infatti effettuare un controllo di validità numerica perde di significato.

Per ogni azione vengono analizzate le precondizioni numeriche relative alla modalità di esecuzione scelta e ne viene controllata la validità all'interno del relativo stato del problema. Al termine di questi controlli vengono verificate anche le condizioni numeriche del goal.

Se un piano è numericamente valido, esso è consistente (perché valido anche a livello simbolico).

(c) **Controllo di configurabilità** Questa operazione anticipa quanto verrà spiegato per la successiva funzionalità di configurazione, pertanto in questo paragrafo ci limitiamo a presentarne gli obiettivi.

Il controllo di configurabilità verifica, per il piano parzialmente specificato fornito, l'esistenza di una configurazione delle modalità di esecuzione delle azioni non specificate che mantenga inalterate le modalità già scelte.

La principale differenza con l'operazione di configurazione di un piano è che in questo caso si effettua solo un controllo di esistenza di una "soluzione", mentre nel processo di configurazione si restituisce un vero e proprio piano completamente istanziato.

Mentre con il controllo di validità numerica l'operatore è in grado di verificare se un piano completamente specificato rispetta tutti i vincoli numerici delle azioni e raggiunge i goal, attraverso il controllo di configurabilità l'operatore può invece verificare se il piano può essere "completato", ovvero se le modalità mancanti possono essere istanziate, in modo da ottenere un piano valido.

5. **Configurazione e riconfigurazione di un piano**

La funzionalità di configurazione (*config*) di un piano è una delle più importanti operazioni messe a disposizione. Grazie ad essa l'operatore ha la possibilità di richiedere al sistema di completare la configurazione delle modalità del piano da lui iniziato.

L'utente può costruire un piano (attraverso le operazioni viste sopra) specificando solamente alcune modalità di esecuzione delle azioni (oppure nessuna), per poi delegare al sistema il completamento (ovvero l'istanziazione), se possibile, di tutte le modalità mancanti.

Il sistema, a partire da un piano parzialmente specificato (cioè con un numero di modalità selezionate minore del numero di azioni del piano), genera, se il piano dato è valido proposizionalmente e se esiste una possibile istanziazione delle modalità mancanti che rispetti tutti i vincoli del problema (controllo di configurabilità), un piano completamente specificato che sia consistente a livello simbolico e a livello numerico.

Per chiarire meglio il concetto supponiamo di aver costruito un piano composto dalle azioni $\mathtt{A}_1$,...,$\mathtt{A}_n$ e di aver assegnato una modalità solamente ad alcune di esse, mentre le modalità delle azioni $\mathtt{A}_i$,$\mathtt{A}_j$,...,$\mathtt{A}_k$ non sono state specificate. Alla richiesta di configurazione del piano, il sistema, dopo aver controllato la consistenza proposizionale del piano, andrà a verificare l'esistenza di una possibile assegnazione per le modalità non specificate tale che

l'intero piano sia consistente a livello numerico. Ovvero l'assegnazione delle modalità dovrà far sì che ogni azione sia consistente numericamente, ossia le sue precondizioni numeriche siano verificate, e che lo stato finale raggiunto dall'applicazione della sequenza di tutte le azioni (con le relative modalità) rispetti i vincoli numerici.

La funzionalità *config* rappresenta il cuore dell'approccio *mixed-initiative* per la pianificazione. L'utente può formulare un piano generico, conoscendo gli obiettivi da raggiungere e il loro ordine, e poi può delegare al sistema la generazione delle modalità di esecuzione delle azioni, operazione solitamente più complessa per via del gran numero di vincoli numerici di cui tener conto. Inoltre se l'utente è a conoscenza di alcune informazioni che lo vincolano ad utilizzare una modalità per una certa azione, grazie a questa funzionalità, può richiedere la configurazione del piano mantenendo inalterata la modalità in questione. Questo può anche essere utile se dopo l'analisi di più piani l'operatore intuisce che una determinata azione vada eseguita con una precisa modalità.

La possibilità di effettuare un'operazione di questo tipo evita all'utente di dover specificare tra i goal particolari vincoli relativi ad azioni precise, semplificandogli notevolmente il lavoro. L'operatore non dovrà far altro che selezionare le modalità desiderate tra quelle possibili e lasciare al sistema la gestione del problema.

Naturalmente non sempre sarà possibile effettuare una configurazione. Maggiori saranno le modalità di esecuzione già selezionate, minori saranno le possibilità di scelta da parte del sistema e minore sarà la probabilità che esista una configurazione possibile. Viceversa un piano completamente non specificato, lascia totale libertà di scelta al sistema, pertanto esso potrà valutare tutte le combinazioni possibili. Tuttavia anche in questo caso non è certo che si possa trovare una configurazione possibile in quanto l'applicabilità di una azione dipende dallo stato del sistema e quest'ultimo è ottenuto dall'applicazione di tutte le azioni precedenti. La configurabilità del piano è quindi dipendente dal tipo e dall'ordine delle azioni che l'utente ha inserito.

La funzionalità di riconfigurazione (*reCon*), invece, si applica ad un piano

completamente specificato (ovvero con tutte le modalità già selezionate). Tale operazione è fornita all'interno della libreria *MMACSP* ([9]) e prevede la ricerca di una nuova configurazione completa del piano. La sua originale applicazione avviene durante l'esecuzione di un piano, nel caso si verifichino contingenze e le risorse numeriche correnti non siano più sufficienti per raggiungere i goal con il piano stabilito, e si applica alle azioni non ancora eseguite (vedi [10]). Nel nostro sistema essa è invece utilizzata *offline* e si suppone che le azioni ancora da eseguire siano tutte quelle inserite nel piano. *reCon* permette dunque di generare una nuova istanza del piano con modalità differenti, eventualmente per confrontare una soluzione con un'altra, senza particolari vincoli di scelta su determinate modalità.

config e *reCon* sono simili a livello implementativo, tuttavia concettualmente si applicano a due tipi di piano differenti. La riconfigurazione si applica ad un piano classico completamente istanziato e valido sia simbolicamente che numericamente. L'operazione di configurazione si applica invece ad un piano parzialmente specificato, ovvero un piano le cui modalità non sono tutte specificate, ed è consistente solo a livello proposizionale (momentaneamente). La consistenza numerica è data infatti dalla valutazione di precondizioni ed effetti di ogni azione, pertanto finché esse non vengono istanziate con le modalità (alle quali sono associati i valori numerici) non è possibile valutare la consistenza numerica del piano.

6. Revise

La funzionalità di *revise* permette all'operatore di modificare le informazioni del problema che influenzano la validità di un piano. Il problema, infatti, contiene le informazioni sullo stato iniziale del mondo e sui *goals* da raggiungere. Modificando tali proprietà, la consistenza di un piano può cambiare completamente, sia a livello proposizionale sia a livello numerico.

Lo stato iniziale del problema infatti determina i valori delle risorse numeriche (e.g la quantità di `fuel`, carburante, iniziale di un `rover`) e le informazioni proposizionali (e.g. la posizione iniziale di un determinato `rover` oppure l'esistenza di una strada tra un sito e un altro). I *goals* invece contengo-

no le condizioni che devono essere verificate affinché il problema sia risolto. Pertanto determinano gli obiettivi che un piano deve raggiungere.

Modificando lo stato iniziale o i *goals*, si genera a tutti gli effetti un nuovo problema. Questo permette all'utente di effettuare delle valutazioni del piano nel caso il problema cambi. Per esempio alterando i *goals* da raggiungere rendendo gli obiettivi numerici più restrittivi (e.g. riducendo il tempo massimo richiesto per completare le operazioni) è possibile analizzare facilmente il comportamento del piano in condizioni più critiche. Ciò permette di sviluppare un piano più robusto in previsione di possibili restrizioni a tempo di esecuzione.

(a) **Revise dello stato iniziale** Per la modifica dello stato iniziale del problema il sistema mette a disposizione operazioni di inserimento, cancellazione ed alterazione dei predicati e dei fluenti numerici in esso contenuti. Per quanto riguarda i predicati, è possibile rimuoverli dallo stato iniziale o aggiungerne di nuovi (specificando gli argomenti, cioè gli oggetti da passare come parametri), mentre la loro modifica consiste nella sostituzione dei parametri con altri. Per quanto riguarda i fluenti numerici, invece, mentre la rimozione e l'inserimento sono analoghi ai predicati, la modifica non consiste nella sostituzione dei parametri ma nell'alterazione del valore numerico associato al fluente stesso.

È bene notare che la rimozione di un predicato o di un fluente numerico comporta la negazione dello stesso. Ovvero, poiché in PDDL si fa una assunzione di mondo chiuso, tutto ciò che non è esplicitato viene trattato come falso. Occorre pertanto prestare attenzione a tale operazione in quanto la rimozione di componenti significative può determinare la non consistenza di un piano o addirittura l'impossibilità di valutarlo.

Se ad esempio, all'interno del dominio dei rover, nella definizione dello stato iniziale di un problema con un unico `rover r1`, rimuoviamo il predicato `in (r1 ...)` senza sostituire tale informazione con un'altra equivalente, un piano che voglia utilizzare una azione di `drive` o di `tp` (le quali hanno tra i prerequisiti la presenza del `rover` in una determinata posizione) non sarà valido proposizionalmente.

Un discorso analogo vale per la rimozione di un fluente numerico, tuttavia in questo caso il sistema non sarà in grado di valutare le precondizioni (o gli effetti) di una azione che utilizzi il fluente rimosso.

Mentre la rimozione di un predicato, dunque, è un'operazione che può avere un significato maggiore ai fini della pianificazione (e.g. vogliamo rimuovere una strada tra un sito e un altro per valutare se esiste una soluzione al problema che non sfrutti tale percorso), la rimozione di un fluente numerico dallo stato iniziale è invece un'operazione da utilizzare solamente in caso di modifica del problema perché non ben formulato. Nello stato iniziale infatti tutti i valori numerici del problema dovrebbero essere inizializzati, pertanto una rimozione che lasci lo stato iniziale senza un valore per un fluente è altamente sconsigliata.

(b) **Revise dei goals**

La modifica dei *goals* è un'operazione molto importante dal punto di vista della pianificazione interattiva. Essa infatti permette, come già accennato, di analizzare il comportamento del piano costruito con vincoli diversi per valutarne la robustezza e la versatilità.

Il sistema mette a disposizione, anche in questo caso, operazioni di modifica, rimozione e inserimento di condizioni di goal. L'utente può inserire un nuovo vincolo proposizionale (e.g `in (r1 l8)` per richiedere che il `rover r1` al termine dell'esecuzione si trovi in `l8`), oppure un vincolo numerico. In questo caso l'operatore può costruire le due espressioni da confrontare attraverso un determinato operatore. I vincoli numerici possono essere "semplici", ovvero vincoli sul valore di un unico fluente numerico (e.g. `(power r1) >= 932.0`), oppure "complessi", ovvero vincoli che confrontano più fluenti numerici relazionati tra di loro attraverso operatori aritmetici (e.g. `((power r1)*(time) /(infoLoss))) <= ((powerC r1) - (memoryC r1))`).

Quelle appena presentate sono le principali funzionalità di alto livello fornite dal sistema sviluppato. È bene notare che la maggior parte di esse sono strettamente correlate tra loro e talvolta i loro significati tendono a sovrapporsi. Per esempio l'inserimento di una azione è una funzionalità che si trova sia in un conte-

sto di costruzione di un piano sia in uno di modifica. Tale operazione assume un diverso significato a livello concettuale (ovvero da una lato fa parte di un processo di realizzazione di un nuovo piano, dall'altro si trova all'interno di un processo di modifica di un piano già esistente), tuttavia a livello pratico esse sono identiche.

Capitolo 6

Implementazione

In questo capitolo presentiamo una descrizione meno concettuale del sistema, soffermandoci maggiormente su dettagli implementativi per fornire al lettore un'idea più concreta del funzionamento e dei servizi messi a disposizione per la pianificazione. Dopo una introduzione all'architettura del sistema e alle librerie in esso utilizzate, verranno mostrate le principali interfacce utente e ne verranno commentati i comportamenti, eventualmente spiegandone i dettagli implementativi. Al termine della presentazione si avrà dunque un'idea più chiara di come il sistema metta a disposizione le funzionalità descritte nella sezione precedente.

Si noti che al fine di mostrare l'intera interfaccia, le figure contenute in questa sezione sono state catturate durante un'esecuzione del software. Pertanto alcune di esse faranno riferimento al dominio dei rover (Appendice A.1), il quale è stato il principale dominio utilizzato durante lo sviluppo.

6.1 Architettura del sistema

L'ambiente è stato sviluppato in Java 1.7. Come già accennato, i domini e i problemi che è in grado di utilizzare devono essere scritti in PDDL 2.1 (livello 2) in modo da supportare le azioni multi-modalità.

Poiché il sistema estende il lavoro di Scala in [9], ne utilizza le librerie PPMajal[1] e MMACSP in esso sviluppate. Inoltre si fa riferimento anche alle librerie

[1]Disponibile su `http://www.di.unito.it/~scala/`

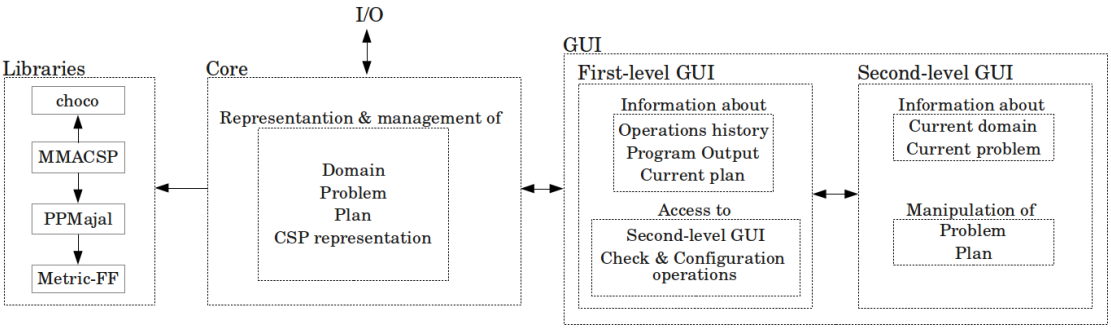


Figura 6.1: L'architettura del sistema

Choco 2.1.4[2], libreria per problemi di soddisfacimento di vincoli (CSP) utilizzata da MMACSP, e Metric-FF([7]), pianificatore utilizzato da PPMajal per la generazione di piani, in quanto, anche se non direttamente richiamate, sono fondamentali per la gestione di problemi con vincoli numerici e per la pianificazione.

In figura 6.1 è stata rappresentata la struttura generale delle componenti del sistema.

Si può notare come esso si sviluppi attorno ad un core implementativo (*Core*, in figura) in cui vengono create, conservate e manipolate le strutture dati degli elementi principali necessari alla pianificazione, ovvero le istanze delle classi del dominio, del problema, del piano e della rappresentazione CSP del piano.

Un primo livello di interfaccia utente (*First-level GUI*, in figura) permette di accedere a tutte le funzionalità del sistema. Da essa è possibile dunque scegliere, attraverso menu a tendina, quali operazioni effettuare. Alcune di esse richiedono un'ulteriore interazione con l'utente, mentre altre possono essere eseguite direttamente (e.g. le operazioni di verifica di consistenza proposizionale o numerica, oppure le operazioni di configurazione e riconfigurazione). Ognuna di esse, dopo l'esecuzione, restituisce delle informazioni all'utente, pertanto a questo livello di interfaccia utente è presente una sezione che raccoglie tali dati, mostrando una vera e propria storia delle operazioni effettuate con i relativi risultati ottenuti. Inoltre viene resa disponibile una visione sempre aggiornata del piano, in modo da avere un'idea chiara del suo stato corrente, e vengono evidenziati gli ultimi elementi modificati.

[2]Per ulteriori informazioni vedi `http://choco.emn.fr`

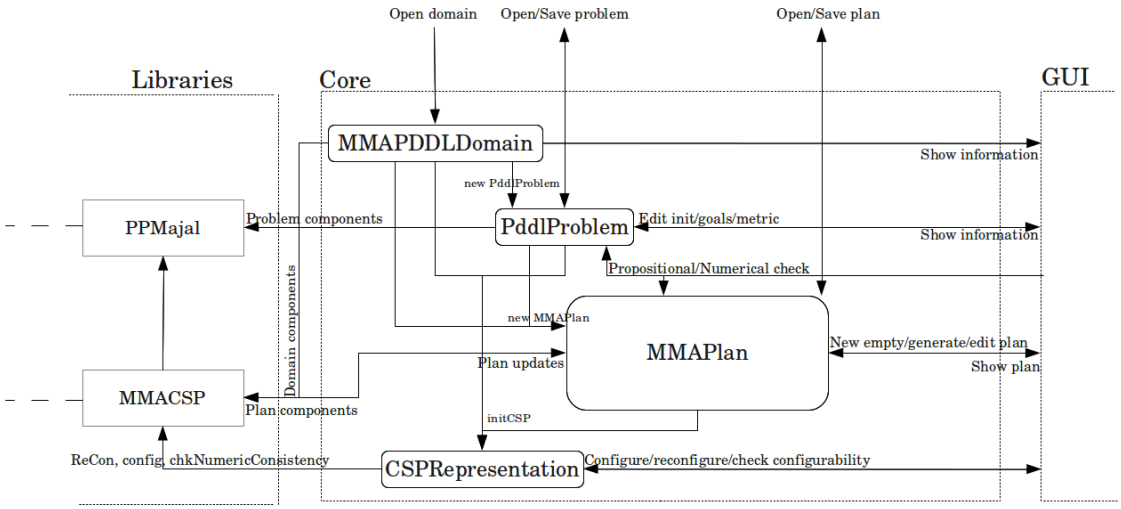


Figura 6.2: Il core del sistema

Scelta l'operazione da effettuare, se necessario, si passa ad un secondo livello di interfaccia utente (*Second-level GUI*, in figura) in cui è possibile entrare nello specifico dell'operazione scelta. In particolare a tale livello si effettuano le procedure di manipolazione del piano e del problema che richiedono un input da parte dell'utente. Tale input può essere la scelta di modifica di un parametro in un altro o l'intenzione di eliminare, aggiungere, sostituire un elemento, o qualsiasi altra possibilità di alterazione o visualizzazione messa a disposizione dal sistema. Questo livello di interfaccia (ad eccezione delle operazioni di semplice visualizzazione di informazioni) agisce su una copia temporanea delle strutture dati e aggiorna il core solamente dopo la conferma delle modifiche da parte dell'utente, riportando quanto effettuato al primo livello, così che esso possa aggiornare la storia delle operazioni.

6.2 Il Core

In figura 6.2 è mostrato un dettaglio del core del sistema. Come si può notare, il piano (istanza della classe *MMAPlan*) è il protagonista principale dell'intera architettura.

Il dominio è un'istanza della classe *MMAPDDLDomain* di MMACSP. Esso deve essere aperto a inizio interazione in quanto indispensabile per la definizione

sia di un problema sia di un piano (e della sua rappresentazione CSP). È pertanto possibile selezionare da file-system un file con estensione .pddl contenente una definizione di dominio.

Il problema è un'istanza della classe *PddlProblem* di PPMajal. Anch'esso è necessario per la definizione di un piano. Inoltre esso fa riferimento ad un dominio. Anche il problema deve essere selezionato da file-system, allo stesso modo del dominio. A differenza di quest'ultimo, tuttavia, sul problema è possibile effettuare delle operazioni di modifica che riguardano lo stato iniziale, i *goals* e la metrica (*Edit init/goals/metric*). È poi anche possibile salvare su file-system il nuovo problema realizzato.

Il sistema mette a disposizione la possibilità di visualizzare le informazioni sia del dominio sia del problema (*Show information*). Tali informazioni conterranno, come si vedrà più in dettaglio nella sezione seguente, dati riguardanti il file a cui il dominio o il problema fanno riferimento, e dati strettamente legati alle due classi, come ad esempio le azioni contenute nel dominio oppure lo stato iniziale del problema.

Al centro del core, come detto, c'è il piano. Esso è un'istanza della classe *MMAPlan* che è istanziabile in riferimento ad un dominio (*MMAPDDLDomain*) e ad un problema (*PddlProblem*), i quali devono dunque essere già stati aperti. Un piano può essere ottenuto dalla creazione di una nuova istanza (*New plan*), dall'apertura di un piano preesistente da file-system (*Open plan*), oppure da una generazione automatica da parte del sistema (*Generate plan*). Quest'ultima operazione di vera e propria pianificazione viene effettuata da PPMajal attraverso Metric-FF, e comporta la sostituzione del piano corrente (se esiste) con un piano completamente nuovo, costruito da capo dal pianificatore.

La classe MMASkeletonPlan Per lo sviluppo del sistema è stata introdotta una nuova classe *MMASkeletonPlan* che estende la classe *MMAPlan* della libreria MMACSP. *MMASkeletonPlan* è la classe centrale per la costruzione interattiva di un piano. Essa permette di rappresentare un piano contenente delle azioni parzialmente specificate, ovvero azioni multi-modalità la cui modalità di esecuzione non è specificata. All'interno del sistema una modalità non specificata viene indicata come *decided by system*. Sarà infatti possibile delegare al sistema la ricerca di una

Codice 6.1: Principali metodi della classe MMASkeletonPlan

```
1  public class MMASkeletonPlan extends MMAPlan{
2    @Override
3    public void putMMAction(
4      String actionName, ActionParametersAsTerms par, String mod);
5    @Override
6    public void parseSolution(String solution_file);
7    @Override
8    public String toString();
9    @Override
10   public boolean savePlan(String path, int start_index);
11 }
```

modalità di esecuzione applicabile, attraverso l'operazione di configurazione di un piano.

Nel codice 6.1 sono mostrate le intestazioni dei principali metodi della classe *MMASkeletonPlan* che sovrascrivono i corrispettivi di *MMAPlan*.

Il metodo *putMMAction* (riga 3) permette l'inserimento di una azione multi-modalità all'interno del piano. Tale azione può essere sia completamente istanziata (ovvero con una modalità di esecuzione specificata, o senza modalità, nel caso l'azione non ne preveda), sia parzialmente istanziata (ovvero senza una modalità di esecuzione specificata nonostante l'azione le preveda). In quest'ultimo caso la chiamata del metodo dovrà contenere come valore del terzo parametro (*mod*, relativo alla modalità) la stringa "decided_by_system".

Il metodo *parseSolution* (riga 6), invece, permette la lettura da file di un piano parzialmente specificato. Per rappresentare all'interno di un file delle azioni parzialmente specificate occorre inserire al posto della modalità la stringa "decided_by_system". In figura 6.3 è mostrato un esempio del contenuto di un file con estensione .pddl rappresentante un piano parzialmente specificato. Esso è costituito da tre azioni, due delle quali sono solo parzialmente specificate (la prima e l'ultima), mentre la seconda (`tp-hr`) è una normale azione multi-modalità completamente istanziata. *parseSolution*, che prende come parametro una stringa rappresentante il *path* del file, ne parsifica il contenuto e costruisce in memoria una istanza di piano parzialmente specificato (*MMASkeletonPlan*) tenendo conto,

```
(drive-decided_by_system r1 m1 m8)
(tp-hr r1 m8)
(comm-decided_by_system r1 m8)
```

Figura 6.3: Semplice esempio di file contenente un piano parzialmente specificato

appunto, delle azioni parzialmente specificate.

Le ultime due funzioni di *MMAPlan* che *MMASkeletonPlan* sovrascrive (righe 8 e 10) permettono la rappresentazione sotto forma di stringa del piano parzialmente specificato e il suo salvataggio su file). Entrambe tengono conto delle modalità non specificate. Nel primo caso una azione parzialmente istanziata viene indicata come "partially instantied" (vedi esempio in figura 6.4), nel secondo caso invece viene trattata allo stesso modo dei due metodi precedenti.

Action Name: drive Parameters: r1 -robot m1 -site m8 -site partially instantied

Figura 6.4: Esempio di stringa rappresentante una azione parzialmente specificata

A seconda che il piano corrente contenga o meno azioni parzialmente specificate, esso viene trattato come *MMASkeletonPlan* o come un normale *MMAPlan*. Su di esso sono eseguibili, indipendentemente dal suo tipo, tutte le possibili modifiche viste nel capitolo 5, quali l'aggiunta, la rimozione di azioni oppure l'alterazione di esse. Il piano può essere modificato dall'utente manualmente attraverso l'interfaccia grafica (*Edit plan*) oppure attraverso richieste di configurazione o riconfigurazione (*Configure/reconfigure*) delegate al CSP (*CSPRepresentation*) e dunque a MMACSP (*config e ReCon*) che si occuperà poi di aggiornare il piano (*Plan updates*). Queste due operazioni di configurazione e riconfigurazione, consistono, come già visto, in una ricerca di una soluzione (o di una soluzione alternativa alla corrente) che associ una modalità di esecuzione ad ogni azione del piano, e possono essere eseguite rispettivamente su un piano parzialmente specificato e su uno completamente specificato.

È inoltre possibile richiedere al sistema il controllo di consistenza proposizionale e/o numerica del piano (*Propositional/Numerical check*). Tale controllo, natural-

mente, farà riferimento anche al problema in quanto dipende dallo stato iniziale e dagli obiettivi da raggiungere.

Check proposizionale Nel codice 6.2 è mostrata, sotto forma di pseudo-codice, una linea guida dell'implementazione del check proposizionale. Si noti come per

Codice 6.2: Pseudo-codice della funzione checkPropositional()

```
1  boolean checkPropositional() {
2    s = Problem.init
3    for each Action a in Plan
4      for each Propositional-Precondition p in a
5        if p is not satisfied in s
6          return false
7      apply a to s
8
9    for each Predicate pred in Problem.goals
10     if pred is not satistied in s
11       return false
12
13   return true
14 }
```

ogni azione venga effettuato un controllo di applicabilità nello stato corrente. Viene cioè verificato che ogni sua precondizione sia soddisfatta nello stato corrente (in caso contrario si ritorna immediatamente poiché l'azione non è applicabile e dunque il piano non è valido). In tal caso viene simulata la sua applicazione, trasformando lo stato corrente in un altro a causa degli effetti dell'azione stessa. Al termine dell'esame di ogni azione si valutano i *goals* nello stato finale ottenuto. Se essi sono tutti soddisfatti allora il piano si ritiene proposizionalmente valido.

L'implementazione vera e propria, nel caso di inconsistenza di una azione, restituisce in output anche informazioni sull'azione e sul predicato non soddisfatto.

Check numerico Nel codice 6.3 è mostrata, sotto forma di pseudo-codice, una linea guida dell'implementazione del check numerico. La sua implementazione è molto simile a quella del check proposizionale. Le principali differenze sono date dal tipo degli elementi su cui viene fatto il test di soddisfacibilità. Nel caso del

Codice 6.3: Pseudo-codice della funzione checkNumeric()

```
boolean checkNumeric() {
  s = Problem.init
  for each Action a in Plan
    mod = a.instantied-modality
    for each Numeric-Precondition p of mod
      if p is not satisfied in s
        return false
    apply a to s

  for each Numeric-Constraint nc in Problem.goals
    if nc is not satistied in s
      return false

  return true
}
```

check proposizionale, infatti, essi sono precondizioni simboliche. Nel caso del check numerico invece sono precondizioni numeriche.

Nell'implementazione, se viene trovata una azione non valida, è riportato in output anche il punto di non validità e vengono mostrati i valori correnti della precondizione non verificata e i valori delle risorse nello stato corrente.

Si noti che il controllo di consistenza numerica come appena descritto è applicabile solamente ad un piano completamente istanziato. Il test di soddisfacibilità di una precondizione numerica di una azione in un determinato stato è effettuabile solamente con azioni la cui modalità è stata specificata. È infatti la modalità di esecuzione che determina precondizioni ed effetti numerici di una azione. Pertanto questa operazione non risulterà abilitata nel caso il piano corrente sia parzialmente specificato.

Per quanto riguarda i piani parzialmente specificati, viene messa a disposizione una funzionalità di controllo di configurabilità (*check configurability*), ovvero un controllo di esistenza di una possibile soluzione al problema.

Infine, la rappresentazione CSP del piano è il quarto principale elemento che costituisce il core del sistema. Essa si costruisce a partire da un dominio, un

problema e un piano e permette l'interpretazione di essi sotto forma di problema di soddisfacimento di vincoli (CSP). La classe *CSPRepresentation* è gestita da MMACSP ed è poi elaborata da Choco (il solver che effettua la ricerca di una soluzione al CSP), e permette la realizzazione delle operazioni di configurazione, riconfigurazione e controllo di consistenza numerica.[3] È bene notare che l'utente non ha un accesso diretto alla rappresentazione CSP. Essa viene utilizzata come struttura di background per le operazioni di configurazione sopracitate ma attraverso l'interfaccia utente non è possibile accedere alla rappresentazione interna. L'interazione con le sue strutture avviene esclusivamente tramite alcune funzioni che permettono di richiedere la costruzione di un CSP e la restituzione dei risultati richiesti, quali le funzioni *init(), reCon(), config() e chkNumericConsistency()*.

La funzione config() Per consentire la gestione di un piano parzialmente specificato, la classe *CSPRepresentation* è stata integrata con una funzione *config()*. Essa permette di effettuare, su un piano parzialmente specificato e proposizionalmente valido, una configurazione di tutte le modalità non istanziate. Tale configurazione, se esiste, assegna a tutte le azioni non ancora istanziate una modalità, generando una vera e propria soluzione al problema. La soluzione costruita, oltre a rispettare tutti i vincoli del problema, mantiene inalterate tutte le modalità già istanziate dall'utente permettendo dunque all'operatore di richiedere la ricerca di una soluzione che completi quella da lui iniziata, senza modificare quanto già stabilito.

La funzione *config()* fa riferimento al solver Choco per la ricerca di una possibile configurazione delle modalità mancanti e riproduce, in parte, le operazioni effettuate dalla funzione *reCon()* di riconfigurazione di un piano, già presente in MMACSP, tenendo conto però delle azioni già istanziate dall'utente. Infatti, mentre per la riconfigurazione il solver ha il compito di trovare una nuova istanziazione di tutte le modalità del piano, nel caso della configurazione si richiede al sistema di cercare una istanziazione di modalità compatibile con quanto fino ad ora deciso dall'operatore, ovvero un insieme di modalità per le azioni parzialmente specificate che completi l'istanziazione parziale corrente.

[3] Per ulteriori dettagli sulla rappresentazione CSP si consultino i testi [9] e [10].

Codice 6.4: Rimozione dal modello CSP dei vincoli sulle modalità di esecuzione delle azioni del piano

```
1 for (int k = i; k < plan.size(); k++) {
2     MMGroundAction action = (MMGroundAction) plan.get(k);
3     Constraint toRemove =
4       (Constraint) this.action2constEqInstantiated.get(action);
5     model.removeConstraint(toRemove);
6 }
```

Il modello del CSP, quando viene inizializzato con i dati di dominio, problema e piano, costruisce una serie di vincoli riguardanti, tra le altre cose, anche le modalità di esecuzione instanziate per le azioni. Per le azioni la cui modalità non è instanziata invece non vengono inseriti nuovi vincoli a riguardo. L'operazione di *reCon()* prevede la rimozione dal modello dei vincoli sulle modalità prima di ricercare una nuova soluzione, poiché ha l'obiettivo di restituire una nuova configurazione delle modalità. L'implementazione di *config()*, invece, poiché si propone di trovare una configurazione esclusivamente per le modalità non specificate, mantenendo inalterate quelle esistenti, lascia all'interno del modello tutti i vincoli sulle modalità già specificate. Per questa ragione, mentre in *reCon()* per ogni azioni contenuta nel piano viene ricercato il vincolo relativo alla sua modalità e viene rimosso (codice 6.4), in *config()* non viene effettuata alcuna operazione di *removeConstraint()* sul modello.

6.3 L'interfaccia utente

In figura 6.5 è mostrata la schermata iniziale del sistema. Come si può notare essa è divisa in tre componenti: il menu principale(1), l'area per la visualizzazione della storia delle operazioni e gli output del programma (2) e l'area per la visualizzazione del piano corrente (3).

Il *Main menu* permette l'accesso a tutte le funzionalità del sistema. Esso è diviso in sei parti, ognuna delle quali permette di accedere ad operazioni riguardanti una determinata area di lavoro. Il sotto-menu *File* dà accesso alle operazioni di salvataggio di piano e problema, e permette di uscire dall'ambiente. Gli altri

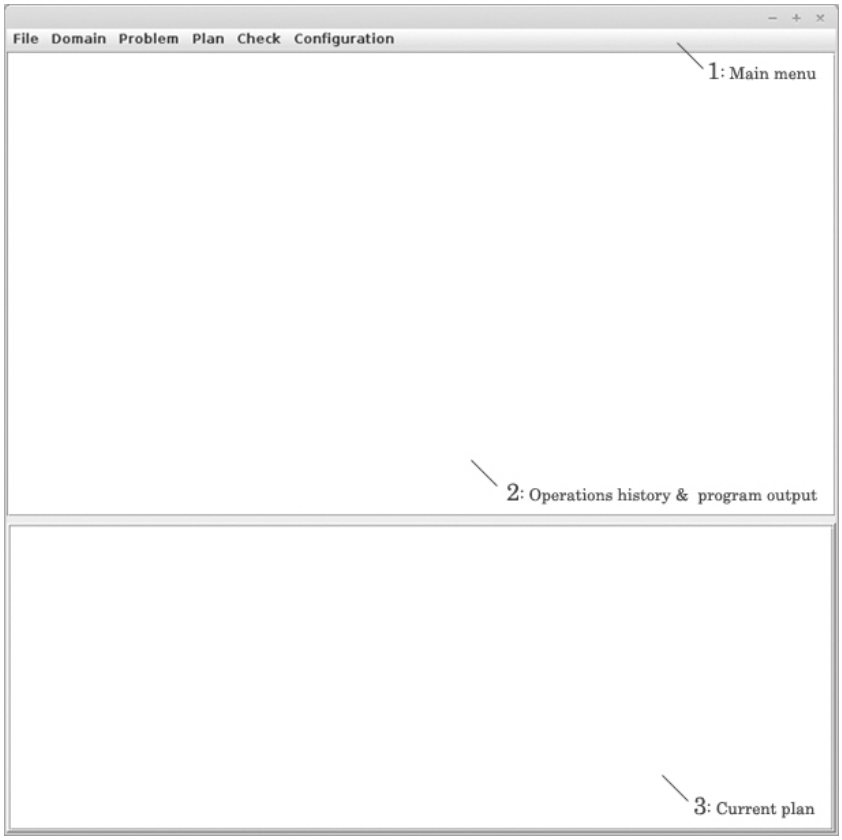


Figura 6.5: First-level GUI

sotto-menu, che verranno descritti meglio nella seguente parte del capitolo, danno accesso alle funzionalità di gestione di dominio, problema e piano e di check e configurazione.

La componente (2) è un'area di testo non editabile e contiene ogni informazione che riguarda le operazioni eseguite o in esecuzione, generando una vera e propria storia delle operazioni. In questo modo l'operatore ha la possibilità di ripercorrere quanto fatto. Inoltre essa contiene anche gli output di alcune delle operazioni effettuabili, come ad esempio il risultato di un test di consistenza proposizionale sul piano con le eventuali motivazioni di fallimento. Pertanto essa fornisce all'utente anche informazioni di tipo più pratico e di necessità più immediata.

Infine la componente (3) è un'area di testo contenente una visualizzazione dello stato corrente del piano. Essa, se cliccata, permette di accedere all'interfaccia di modifica del piano (vedi 6.15).

Entrambe le aree di testo vengono mantenute costantemente aggiornate da ognuna delle operazioni che verranno presentate di seguito, in modo da mostrare all'utente una visione sempre attuale dello stato del sistema e delle strutture dati.

6.3.1 Gestione del dominio

Figura 6.6: Menu per la gestione del dominio

Il menu *Domain* (figura 6.6) dà accesso alle funzionalità di gestione del dominio offerte dal sistema.

Il primo elemento in esso contenuto (1, *Open domain...*) permette di aprire un file con estensione .pddl contenente la definizione di un dominio e di creare una nuova istanza della classe *MMAPDDLDomain* che, come visto a inizio capitolo, costituisce la rappresentazione interna del dominio stesso.

Il secondo, invece, (2, *Show informations*) dà accesso ad una nuova finestra (mostrata in figura 6.7) che consente la visualizzazione di tutte le informazioni riguardanti il dominio corrente. Attraverso questa interfaccia è dunque possibile visualizzare sia informazioni riguardanti il file di dominio aperto (e.g. la sua posizione all'interno del file system), sia informazioni riguardanti le sue caratteristiche. Vengono infatti mostrati, oltre ai requisiti del dominio e ai tipi utilizzati, anche i

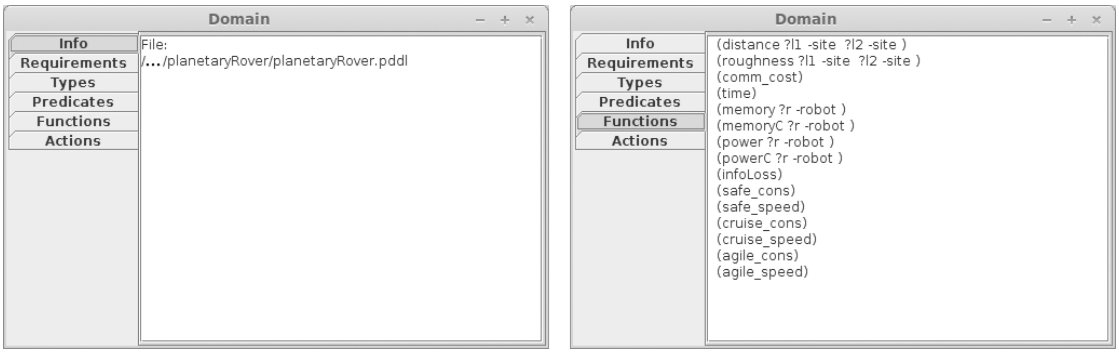


Figura 6.7: Due esempi di informazioni sul dominio visualizzate

predicati, le funzioni e le azioni in esso presenti. Ovvero vengono mostrate tutte le componenti caratterizzanti di un dominio, in modo da dare all'utente una visione chiara di quanto e come esso sia articolato.

Nell'ambiente sviluppato, il dominio non è stato reso modificabile in quanto tale operazione rientra solamente in parte negli obiettivi della tesi. Le informazioni su di essono possono essere visualizzate in qualunque momento dell'esecuzione del sistema, a seconda delle necessità dell'operatore e rimarranno inalterate. È però possibile aprire, durante l'esecuzione, un nuovo dominio (1). Naturalmente questa operazione comporta la cancellazione del problema e del piano correnti e dunque la perdita di tutte le modifiche non salvate. Tuttavia in questo modo è possibile utilizzare un dominio diverso.

6.3.2 Gestione del problema

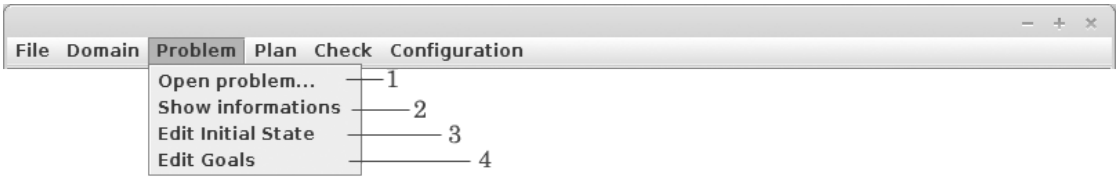


Figura 6.8: Menu per la gestione del problema

Il menu *Problem* (figura 6.8) dà accesso alle funzionalità di gestione del problema offerte dal sistema.

L'elemento (1), permette, analogamente al dominio, di aprire un file con estensione .pddl contenente la definizione di un problema e di creare una nuova istanza della classe *PddlProblem*. Naturalmente il problema selezionato deve essere compatibile con il dominio corrente.

Anche in questo caso è possibile effettuare tale operazione in qualunque momento dell'esecuzione, con la conseguente cancellazione del piano corrente. Il dominio tuttavia in questo caso rimane inalterato. Per un determinato dominio infatti normalmente si definiscono più problemi.

Anche per quanto riguarda il problema è possibile richiedere la visualizzazione delle sue informazioni (2). Come mostrato in figura 6.9 ciò da accesso, come per il dominio, sia ad informazioni di carattere generale (e.g il nome del problema e

del dominio di riferimento o il percorso del file corrente su file-system) sia a tutte le informazioni legate alla pianificazione, ovvero lo stato iniziale del problema, i *goals*, le sue funzioni, la metrica specificata e gli oggetti in esso contenuti.

Al fine di favorire la pianificazione interattiva, il problema in esame è stato reso modificabile. Tale modifica permette infatti di valutare un piano in condizioni diverse. Il problema infatti contiene lo stato iniziale del mondo e gli obiettivi che si intende raggiungere. Al variare di queste informazioni cambia anche la validità di un piano e l'utilizzo delle sue risorse (vedi considerazioni fatte nel capitolo 5 per ulteriori informazioni su conseguenze e vantaggi della modifica del problema).

Modifica dello stato iniziale La modifica dello stato iniziale del problema (3) è resa possibile dall'interfaccia mostrata in figura 6.10. Come si può notare, in essa viene mostrato un elenco di tutte le informazioni che il problema fornisce sullo stato iniziale (a), ossia tutti i predicati che sono veri (ricordiamo l'assunzione di mondo chiuso in PDDL per cui ogni informazione non specificata è considerata falsa) e tutti i valori iniziali dei fluenti numerici. Per ogni predicato o fluente numerico sono messe a disposizione, attraverso due pulsanti, un'operazione di modifica (b) e una di eliminazione (c). L'operazione di modifica (*edit*) di un predicato, permette di cambiarne gli oggetti da esso utilizzati come parametri attuali (figura 6.11a). La modifica di un fluente numerico, invece, (figura 6.11b) consiste nella sostituzione del suo valore con un altro.

Per facilitare la ricerca di un elemento all'interno dello stato iniziale (poiché possono esservi migliaia di elementi, a seconda della complessità del dominio e

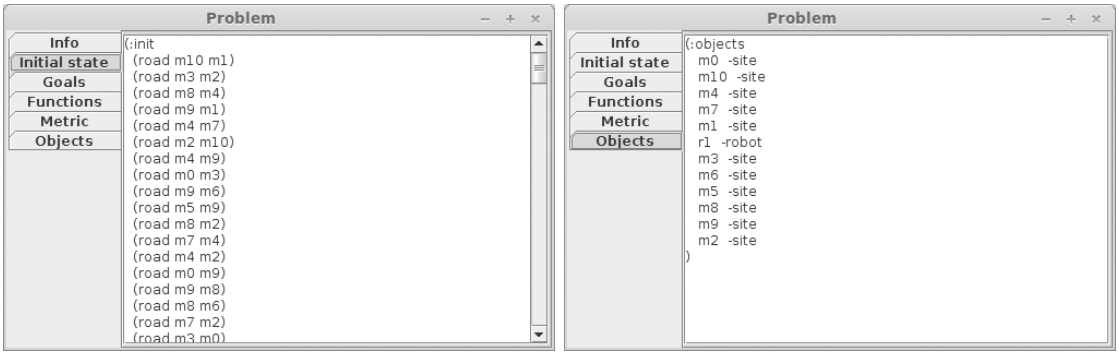


Figura 6.9: Due esempi di informazioni sul problema visualizzate

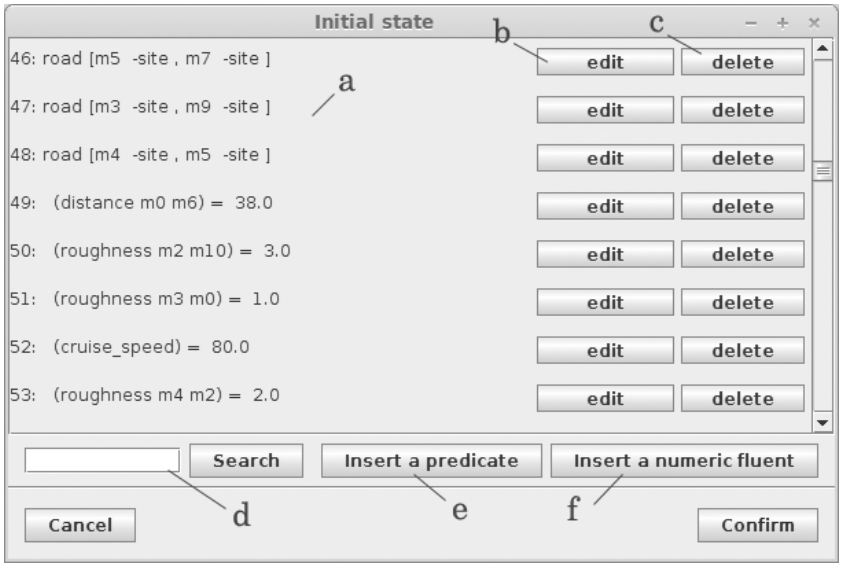


Figura 6.10: Interfaccia per la modifica dello stato iniziale del problema

del problema), è stata introdotta una funzionalità di ricerca (d) che permette di trovare facilmente un elemento specificando il suo nome, i suoi parametri o il suo valore.

Naturalmente oltre a modificare gli elementi già esistenti è anche possibile inserirne di nuovi. Anche in questo caso è possibile inserire sia un predicato (e) sia un fluente numerico (f). L'interfaccia utente per l'inserimento di un nuovo elemento è analoga a quella della modifica, con la differenza che, nel caso del predicato, è consentita anche la scelta di un predicato tra tutti quelli presenti nel dominio corrente, e nel caso del fluente numerico, la scelta di un fluente tra tutte le funzioni del dominio e degli oggetti da assegnare come parametri, oltre al suo valore.

Modifica dei goals La modifica dei *goals* del problema (4) è possibile attraverso l'interfaccia mostrata in figura 6.12. Essa è impostata analogamente all'interfaccia relativa alla modifica dello stato iniziale. Viene dunque mostrato un elenco di elementi rappresentanti gli obiettivi del problema (a), ognuno dei quali è modificabile(b) o rimovibile (c).

Viene inoltre messa a disposizione la possibilità di inserire nuovi *goals*, i quali possono essere vincoli simbolici (predicati che devono essere veri al termine dell'e-

(a) Modifica di un predicato (b) Modifica di un fluente numerico

Figura 6.11: Interfacce per la modifica degli elementi dello stato iniziale

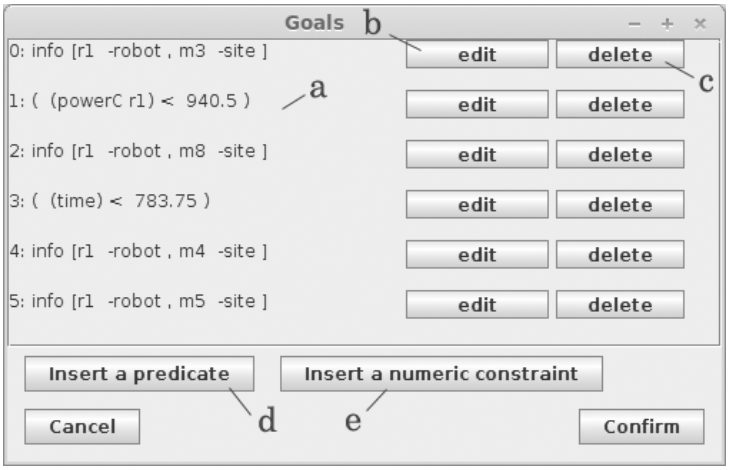


Figura 6.12: Interfaccia per la modifica dei goals del problema

secuzione del piano) oppure vincoli numerici (vincoli sui valori finali delle risorse del dominio). Nel primo caso (d) l'interfaccia utente è la stessa fornita per l'aggiunta di un predicato allo stato iniziale (vedi 6.11a). Nel secondo caso (e), invece, l'interfaccia utente è quella mostrata in figura 6.13.

I vincoli numerici sono delle comparazioni tra due espressioni. Tali espressioni possono essere dei valori numerici semplici, dei fluenti numerici oppure delle espressioni complesse formate a loro volta da due espressioni messe in relazione tra loro attraverso un operatore aritmetico. Usando l'interfaccia utente per l'inserimento di un vincolo numerico è possibile costruire le due espressioni da confrontare e scegliere l'operatore di confronto appropriato.

Un'interfaccia analoga viene richiamata per la costruzione di una espressione,

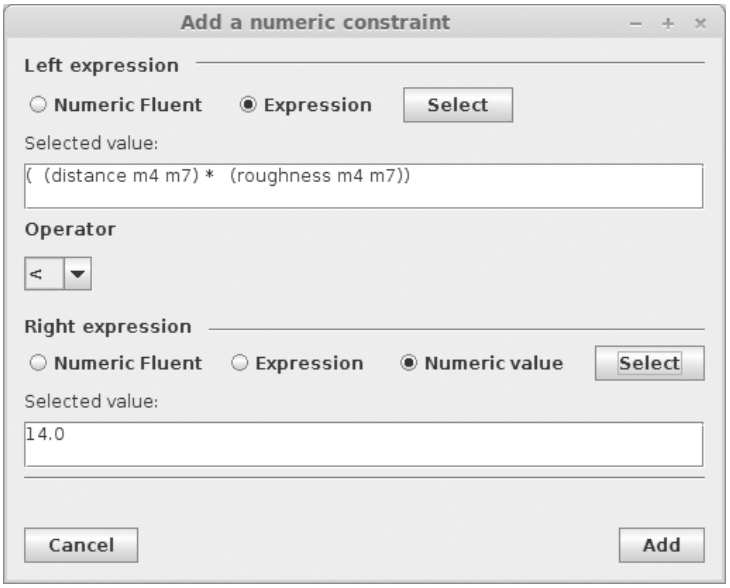


Figura 6.13: Interfaccia per l'inserimento di un vincolo numerico

con la differenza che al suo interno gli operatori sono aritmetici (`+`, `*`, `-`, `/`) e non di confronto (`<`, `<=`, `=`, `>=`, `>`).

6.3.3 Gestione del piano

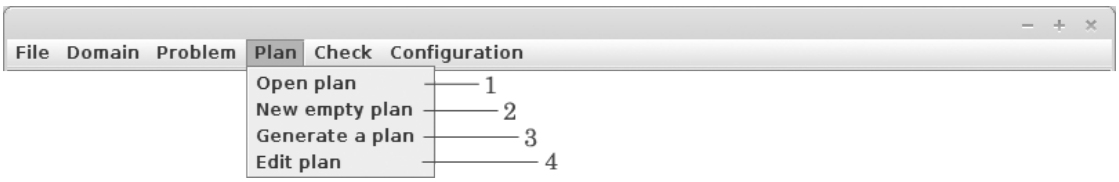


Figura 6.14: Menu per la gestione del piano

Il menu *Plan* (figura 6.14) dà accesso alle funzionalità di gestione del piano offerte dal sistema.

Analogamente al problema e al dominio, l'elemento (1) permette di aprire un file con estensione .pddl contenente la definizione di un piano e di creare una nuova istanza della classe *MMAPlan*. Se il piano selezionato è un piano parzialmente specificato, esso verrà automaticamente interpretato come istanza della classe *MMASkeletonPlan* e, in base al tipo di piano aperto, verranno abilitate le appropriate funzionalità di configurazione/riconfigurazione e di controllo di consistenza

numerica/controllo di configurabilità. In seguito all'apertura del piano sarà possibile effettuare su di esso qualsiasi modifica. Il file originale, comunque, non verrà alterato fino ad un eventuale salvataggio del nuovo piano da parte dell'utente.

L'elemento (2) permette di creare un nuovo piano vuoto. In seguito all'esecuzione di questa operazione verrà aperta la finestra di modifica del piano (6.15) per permettere l'aggiunta di nuove azioni al piano.

L'elemento (3), invece, permette di richiedere al panificatore interno la generazione di un piano per il problema corrente, se esiste. L'eventuale piano ottenuto sarà valido sia a livello proposizionale sia a livello numerico. Dopo la pianificazione sarà possibile modificare il risultato secondo le proprie necessità. È bene ricordare, però, che la generazione automatica di un piano comporta la perdita del piano corrente, infatti essa non tiene conto del piano già sviluppato, ma ne genera uno nuovo, indipendente dal primo.

Questa operazione si discosta leggermente dal concetto di pianificazione interattiva, se utilizzata come unica risorsa del sistema. Se invece si sfrutta questa funzionalità come un inizio da cui partire per la realizzazione di un piano più personalizzato ricercando una maggiore robustezza e versatilità, può risultare una risorsa molto utile.

Modifica del piano L'elemento (4), infine, permette di accedere all'interfaccia di modifica del piano. In figura 6.15 è mostrata tale interfaccia. Come si può

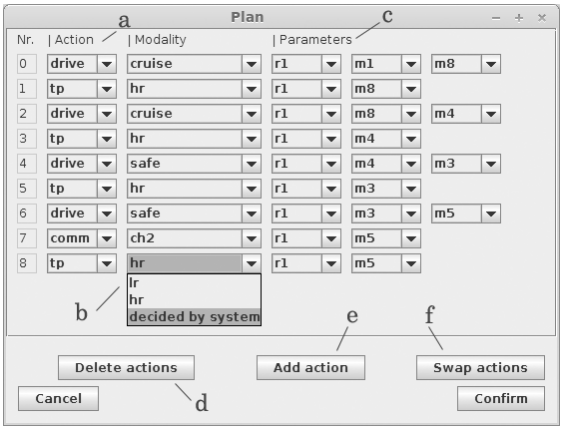


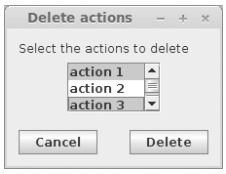
Figura 6.15: Interfaccia per la modifica del piano

notare essa si sviluppa principalmente su tre colonne che contengono le azioni del

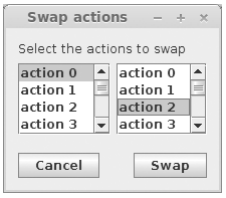
piano (a), le modalità ad esse associate (b) e i parametri scelti (c). Per ogni azione le modalità selezionabili sono tutte quelle definite nello schema dell'azione stessa, con l'aggiunta della scelta *decided by system* che, come già visto, permette di delegare al sistema la decisione della modalità da applicare. Per quanto riguarda i parametri, invece, essi sono selezionabili dall'elenco degli oggetti contenuti nel problema che hanno un tipo compatibile con quello del parametro stesso.

Si noti che l'impossibilità di inserire valori non corretti da un punto di vista sintattico non implica la validità e l'applicabilità di una azione. Tali controlli sono effettuabili invece attraverso le operazioni di check, solamente dopo aver confermato le modifiche del piano.

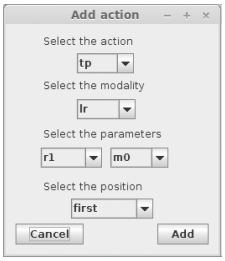
All'interno dell'interfaccia sono presenti anche dei pulsanti che danno accesso alle operazioni di rimozione di azioni (d), inserimento di una nuova azione (e) e scambio di posizione di due azioni (f). Le finestre relative a queste operazioni sono mostrate in figura 6.16. La finestra di inserimento di una nuova azione (6.16c)



(a) Rimozione di azioni



(b) Scambio di posizione di due azioni



(c) Inserimento di una azione

Figura 6.16: Interfacce per la modifica degli elementi del piano

permette di selezionare una azione a partire da tutte quelle presenti nella definizione del dominio. Inoltre dà la possibilità di selezionare la modalità di esecuzione o di scegliere di lasciare la decisione al sistema, attraverso l'opzione *decided by system*. Oltre agli oggetti da passare come argomenti, infine, è possibile decidere

la posizione di inserimento dell'azione all'interno del piano (*first, between action 0 and 1, ..., last*).

6.3.4 Operazioni di check

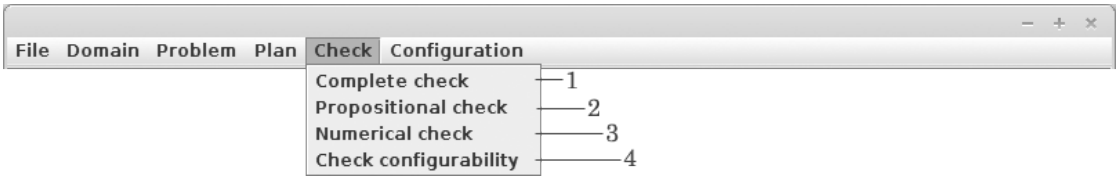


Figura 6.17: Menu per la gestione delle operazioni di check

Il menu *Check* (figura 6.17) dà accesso alle funzionalità di controllo di consistenza (o di configurabilità) del piano corrente.

Tali operazioni non sono correlate di interfaccia grafica in quanto non richiedono ulteriori input da parte dell'utente. Esse comunque restituiscono un risultato nell'area di testo (2) mostrata in figura 6.5. Nessuna delle operazioni di check apporta delle modifiche alle strutture dati correnti. Esse si limitano ad analizzarle e a restituire all'operatore un output che contenga le informazioni richieste.

La prima operazione possibile (1) è un'operazione di controllo completo di consistenza, ovvero un controllo di consistenza del piano sia a livello proposizionale sia a livello numerico. Un possibile output fornito dal sistema può essere quello contenuto in figura 6.18. L'operazione di check è stata effettuata su un piano completamente specificato[4] composto da nove azioni, proposizionalmente valido ma non valido numericamente.

L'operazione di check completo effettua prima una verifica di consistenza proposizionale e, solamente se il piano è valido a livello simbolico, viene fatta anche la verifica a livello numerico. Come si può notare dalla figura, per ogni azione

[4]Le operazioni (3) e (4) vengono attivate in maniera esclusiva in base al tipo del piano corrente. Se esso è un piano completamente specificato, viene attivata l'operazione di check numerico. Se il piano è invece parzialmente specificato, viene attivata l'operazione di controllo di configurabilità. Poiché il piano utilizzato per l'esempio è un piano completamente specificato, per il test numerico è stata effettuata un'operazione di controllo di consistenza numerica. Se il piano fosse stato istanza di *MMASkeletonPlan*, ovvero se fosse stato un piano parzialmente specificato, sarebbe stato effettuato, al suo posto, il controllo di configurabilità.

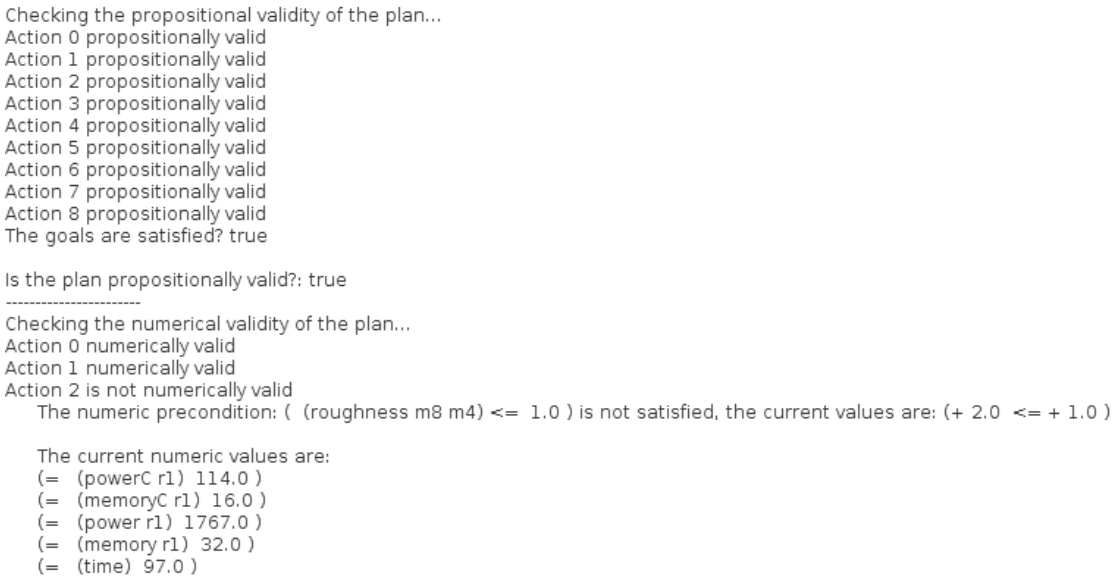


Figura 6.18: Output dell'operazione di Complete check su un piano valido solo simbolicamente

viene riportata la sua (non)validità, con i relativi dettagli riguardanti i valori degli elementi del piano nel punto di non validità.

L'operazione (2) consiste nel controllo di consistenza proposizionale del piano. Essa viene richiamata dall'operazione (1) come primo passaggio e restituisce un valore booleano che indica la validità simbolica (o meno) del piano. E' comunque possibile effettuare esclusivamente questo controllo per verificare solo la consistenza proposizionale di un piano. L'output di questa operazione consiste in un elenco di informazioni riguardo tutte le azioni del piano. Nel caso di una azione consistente a livello proposizionale viene semplicemente indicata come valida, mentre nel caso di inconsistenza viene indicato il primo predicato contenuto nelle precondizioni che invece nello stato corrente non è presente. Inoltre l'output contiene anche informazioni sugli obiettivi simbolici del problema, indicando se sono o meno verificati.

L'operazione (3) consiste invece nel controllo di consistenza numerica di un piano completamente specificato. Essa restituisce un valore booleano che indica la validità numerica (o meno) del piano, ed è un'operazione effettuabile solamente su un piano proposizionalmente valido. Anche in questo caso l'output è un elenco

di informazioni sulle azioni del piano. Nel caso di una azione consistente a livello numerico essa viene indicata come valida, mentre nel caso di inconsistenza viene indicata la precondizione numerica non valida e il suo valore corrente. Inoltre l'output contiene anche informazioni sugli obiettivi numerici del problema, indicando se sono o meno verificati e i valori delle risorse numeriche nell'ultimo stato raggiunto.

L'ultima operazione di check (4) consiste nel controllo di esistenza di una possibile configurazione di modalità di esecuzione per il piano corrente. Tale operazione si applica esclusivamente ad un piano parzialmente specificato e, utilizzando il solver CSP Choco, cerca una soluzione che rispetti le decisioni prese dall'utente e completi le modalità mancanti. Nella ricerca dell'esistenza di una possibile configurazione viene tenuto conto delle modalità già specificate dall'utente, le quali vengono considerate come dei vincoli da rispettare.

Si ricorda che le operazioni di check non effettuano modifiche al piano, pertanto anche questa operazione serve unicamente ad accertarsi dell'esistenza di almeno una possibile soluzione al problema che contenga quanto già deciso, tuttavia restituisce esclusivamente una risposta booleana. Questo permette all'utente che sta costruendo un piano di accertarsi man mano che quanto costruito sia consistente sia a livello proposizionale sia a livello numerico e che non si proceda a pianificare su una strada impraticabile.

6.3.5 ffigurazione e riconfigurazione

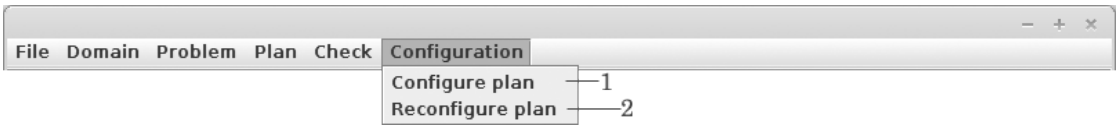


Figura 6.19: Menu per la gestione delle operazioni di configurazione

Il menu *Configuration* (figura 6.19) dà accesso alle funzionalità di configurazione (o riconfigurazione) del piano corrente.

Tali operazioni, come quelle di check, non sono correlate di interfaccia grafica in quanto non richiedono ulteriori input da parte dell'utente. Esse oltre a restituire un risultato nell'area di testo (2) mostrata in figura 6.5, aggiornano anche la

visualizzazione del piano (area di testo (3) in figura 6.5) evidenziando le modalità appena configurate. Tali operazioni agiscono sulla struttura del piano andando a modificare le modalità di esecuzione delle azioni in esso contenute.

L'operazione (1) è abilitata solamente nel caso il piano corrente sia parzialmente specificato ed è l'operazione che richiama la funzione *config()* vista nel paragrafo 6.2. In figura 6.20 è mostrato un esempio di visualizzazione di un piano dopo una

```
0: Action Name:drive Parameters: r1 -robot m1 -site m8 -site mod:safe
1: Action Name:tp Parameters: r1 -robot m8 -site mod:hr
2: Action Name:drive Parameters: r1 -robot m8 -site m4 -site mod:safe
3: Action Name:tp Parameters: r1 -robot m4 -site mod:hr
4: Action Name:drive Parameters: r1 -robot m4 -site m3 -site mod:safe
5: Action Name:tp Parameters: r1 -robot m3 -site mod:hr
6: Action Name:drive Parameters: r1 -robot m3 -site m5 -site mod:safe
7: Action Name:comm Parameters: r1 -robot m5 -site mod:ch2
8: Action Name:tp Parameters: r1 -robot m5 -site mod:hr
```

Figura 6.20: Esempio di visualizzazione di un piano appena configurato

configurazione. Il piano in questione è composto da 9 azioni. Per tutte le azioni di *drive* la modalità non era stata specificata, mentre per le altre sì. L'operazione di configurazione ha trovato una configurazione di tutte le modalità mancanti, rispettando i vincoli delle altre. Nella rappresentazione presentata all'utente vengono evidenziate le azioni le cui modalità sono appena state configurate, in modo da rendere più immediata la visualizzazione delle modifiche.

L'operazione (2) è invece abilitata solamente per un piano completamente specificato e richiama la funzione *reCon()* di MMACSP. L'output di questa operazione è analogo a quello mostrato in figura 6.20 per la configurazione, con la differenza che in questo caso vengono evidenziate le azioni le cui modalità sono state riconfigurate.

Capitolo 7

Scenari di utilizzo

In questo capitolo verranno presentati tre scenari di utilizzo del sistema al fine di mostrarne le potenzialità. Verrà evidenziata l'utilità dell'approccio mixed-initiative adottato e alcuni dei vantaggi che si possono ottenere dal suo utilizzo. I primi due scenari mostrano come possa essere adattato in modo efficiente un piano dopo l'inserimento di nuovi *goals*. Nello scenario 7.1 i nuovi goal sono di tipo proposizionale. Viene dunque mostrato l'adattamento di un piano all'aggiunta di nuovi obiettivi da raggiungere. Nello scenario 7.2, invece, i nuovi *goals* sono di tipo numerico. Pertanto sono mostrate le possibilità messe a disposizione dell'operatore per raffinare il piano in base ai nuovi requisiti numerici. Lo scenario 7.3, infine, mostra la funzionalità di configurazione di un piano parzialmente specificato.

7.1 Adattamento piano (Nuovi Obiettivi)

Il primo scenario mostra come sia possibile adattare in maniera efficiente un piano a nuovi obiettivi simbolici. Grazie alle conoscenze dell'operatore umano e alla sua consapevolezza del dominio, insieme agli strumenti forniti dal sistema, sarà infatti possibile costruire una nuova soluzione valida (adattando quella corrente) senza dover effettuare una ri-pianificazione da zero.

Lo scenario è presentato sul dominio dei rover (Appendice A.1), in cui vi sono 3 azioni diverse: `drive`, che ha lo scopo di muovere un rover da un sito ad un altro e che può essere eseguita in 3 modalità diverse (`safe, cruise, speed`) a seconda

```
(= (power r1) 4131.0)
(= (powerC r1) 0)
(= (time) 0)
(= (memory r1) 160)
(= (memoryC r1) 0)
(= (infoLoss) 0)
(= (comm_cost) 0)
```

(a) Stato iniziale dei fluenti numerici variabili del problema

```
(info r1 m12)
(info r1 m11)
(info r1 m2)
(info r1 m25)
(info r1 m21)
(info r1 m16)
(info r1 m7)
(info r1 m17)
(info r1 m9)
(info r1 m4)
(< (powerC r1) 1651.65)
(< (time) 2307.9)
(< (memoryC r1) 1)
(< (comm_cost) 2 )
```

(b) I *goals* del problema

Figura 7.1: Stato initiale e *goals* del problema di esempio

del `power` disponibile sul rover e della rugosità del percorso da effettuare, `tp`, che permette ad un rover di scattare una fotografia di un certo sito e che può essere eseguita in 2 modalità (`lr, hr`) in base alla memoria a disposizione, e `comm` che permette al rover di comunicare quanto ha memorizzato e che può essere eseguita nelle due modalità `ch1, ch2` in base alla quantità di `power` del rover.

Supponiamo di avere un problema il cui stato iniziale è in parte mostrato in figura 7.1a (sono mostrati per chiarezza solamente i fluenti numerici i cui valori possono subire variazioni durante l'esecuzione di un piano) e i cui *goals* iniziali sono mostrati in figura 7.1b. Supponiamo inoltre di essere a conoscenza di un piano in grado di risolvere tale problema, ovvero un piano (7.2) consistente sia a livello proposizionale sia a livello numerico che raggiunge tutti gli obiettivi rispettando ogni vincolo, come mostrato dagli output delle funzioni di check proposizionale e check numerico applicate su di esso (figura 7.3). L'operatore può essere interessato ad ampliare il piano corrente, inserendo nuovi obiettivi da raggiungere. Per esempio potrebbe ritenere utile aggiungere agli obiettivi la richiesta di catturare delle informazioni anche dei due siti `m6` e `m14`. Per fare ciò aggiungiamo ai *goals* i vincoli `(info r1 m6)` e `(info r1 m14)`.

In seguito a tale operazione, tuttavia, il piano attuale non risulta più essere valido proposizionalmente. I due nuovi vincoli infatti (come mostrato in figura 7.4) non sono soddisfatti.

```
(TP-HR R1 M9)
(DRIVE-AGILE R1 M9 M21)
(DRIVE-SAFE R1 M21 M25)
(TP-HR R1 M25)
(DRIVE-SAFE R1 M25 M21)
(TP-HR R1 M21)
(DRIVE-SAFE R1 M21 M14)
(DRIVE-SAFE R1 M14 M17)
(TP-LR R1 M17)
(DRIVE-SAFE R1 M17 M16)
(TP-LR R1 M16)
(DRIVE-SAFE R1 M16 M11)
(TP-HR R1 M11)
(DRIVE-SAFE R1 M11 M6)
(DRIVE-SAFE R1 M6 M2)
(TP-HR R1 M2)
(DRIVE-CRUISE R1 M2 M5)
(DRIVE-SAFE R1 M5 M12)
(TP-LR R1 M12)
(DRIVE-SAFE R1 M12 M5)
(DRIVE-SAFE R1 M5 M7)
(TP-LR R1 M7)
(DRIVE-SAFE R1 M7 M6)
(DRIVE-AGILE R1 M6 M5)
(DRIVE-SAFE R1 M5 M4)
(TP-LR R1 M4)
(COMM-CH1 R1 M4)
```

Figura 7.2: Un piano valido per il problema 7.1

```
...
Action 24 propositionally valid
Action 25 propositionally valid
Action 26 propositionally valid
The goals are satisfied? true
Is the plan propositionally valid?: true
```

(a) Check proposizionale

```
...
Action 24 numerically valid
Action 25 numerically valid
Action 26 numerically valid
Goals satisfied? true
```

(b) Check numerico

Figura 7.3: Output del check completo sul piano 7.2

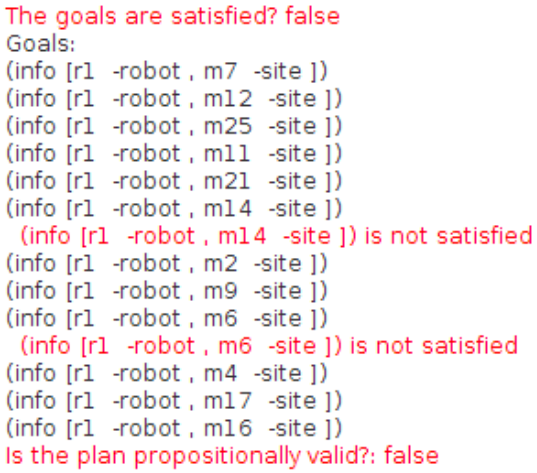
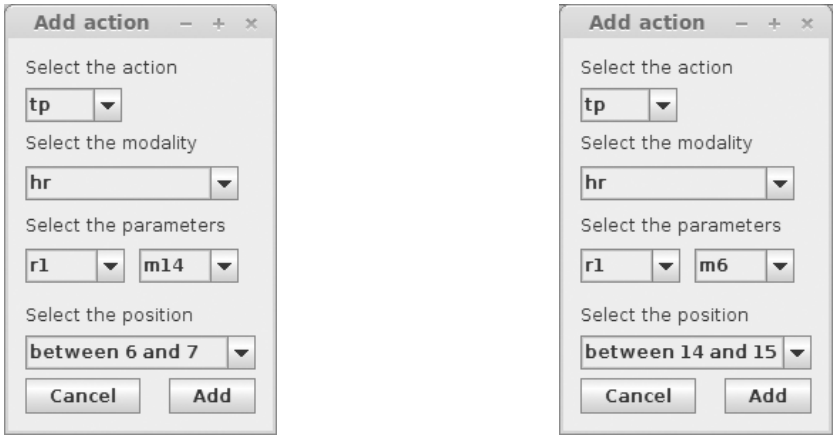


Figura 7.4: Check proposizionale sul nuovo piano

L'approccio mixed-initiative adottato dal sistema, oltre ad aver permesso l'alterazione dei *goals*, dà, in questo caso, la possibilità di aggiungere manualmente le due azioni necessarie al raggiungimento dei nuovi obiettivi. L'operatore può infatti inserire due azioni di `tp` nei punti del piano in cui sa (perché lo deduce dal piano stesso) che il rover si trovi nella posizione corretta (vedi figura 7.5). In questo modo grazie a due semplici operazioni manuali abbiamo ottenuto un piano nuovamente valido a livello simbolico, evitando di dover richiedere al pianificatore una ri-pianificazione da zero.

Tuttavia l'inserimento delle due nuove azioni nel piano ha degli impatti negativi sulle risorse numeriche. Come si può notare dalla figura 7.6 (ottenuta effettuando nuovamente un check numerico sul piano appena modificato) i valori finali del fluente *time* e del fluente *powerC* non rispettano più i vincoli definiti nei *goals*. Poiché spesso per quanto riguarda i valori numerici non è banale determinarne cause ed effetti, l'operatore a questo punto può decidere di richiedere al sistema di cercare automaticamente una riconfigurazione (operazione *Reconfigure*) delle modalità di esecuzione delle azioni del piano che lo renda numericamente consistente. In figura 7.7 è mostrato il risultato di questa operazione con evidenziate le azioni le cui modalità sono state modificate.

È bene notare che anche questa operazione di riconfigurazione ha richiesto pochi



(a) TP R1 M14 (b) TP R1 M6

Figura 7.5: Inserimento azioni di TP nel piano 7.2

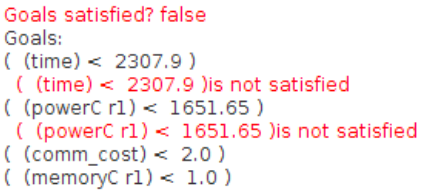


Figura 7.6: Check numerico sul nuovo piano

0: Action Name:tp Parameters: r1 -robot m9 -site **mod:lr**
1: Action Name:drive Parameters: r1 -robot m9 -site m21 -site mod:agile
2: Action Name:drive Parameters: r1 -robot m21 -site m25 -site mod:safe
3: Action Name:tp Parameters: r1 -robot m25 -site **mod:lr**
4: Action Name:drive Parameters: r1 -robot m25 -site m21 -site **mod:agile**
5: Action Name:tp Parameters: r1 -robot m21 -site **mod:lr**
6: Action Name:drive Parameters: r1 -robot m21 -site m14 -site **mod:agile**
7: Action Name:tp Parameters: r1 -robot m14 -site **mod:lr**
8: Action Name:drive Parameters: r1 -robot m14 -site m17 -site mod:safe
9: Action Name:tp Parameters: r1 -robot m17 -site mod:lr
10: Action Name:drive Parameters: r1 -robot m17 -site m16 -site mod:safe
11: Action Name:tp Parameters: r1 -robot m16 -site mod:lr
12: Action Name:drive Parameters: r1 -robot m16 -site m11 -site mod:safe
13: Action Name:tp Parameters: r1 -robot m11 -site **mod:lr**
14: Action Name:drive Parameters: r1 -robot m11 -site m6 -site mod:safe
15: Action Name:tp Parameters: r1 -robot m6 -site **mod:lr**
16: Action Name:drive Parameters: r1 -robot m6 -site m2 -site mod:safe
17: Action Name:tp Parameters: r1 -robot m2 -site **mod:lr**
18: Action Name:drive Parameters: r1 -robot m2 -site m5 -site **mod:safe**
19: Action Name:drive Parameters: r1 -robot m5 -site m12 -site mod:safe
20: Action Name:tp Parameters: r1 -robot m12 -site mod:lr
21: Action Name:drive Parameters: r1 -robot m12 -site m5 -site **mod:agile**
22: Action Name:drive Parameters: r1 -robot m5 -site m7 -site **mod:agile**
23: Action Name:tp Parameters: r1 -robot m7 -site mod:lr
24: Action Name:drive Parameters: r1 -robot m7 -site m6 -site mod:safe
25: Action Name:drive Parameters: r1 -robot m6 -site m5 -site **mod:safe**
26: Action Name:drive Parameters: r1 -robot m5 -site m4 -site mod:safe
27: Action Name:tp Parameters: r1 -robot m4 -site mod:lr
28: Action Name:comm Parameters: r1 -robot m4 -site mod:ch1

Figura 7.7: Risultato dell'operazione di riconfigurazione del piano

secondi (3.8 secondi [1]). Una eventuale richiesta di ri-pianificazione al pianificatore Metric-FF, invece, dopo un tempo pari a tre ore (10800 secondi)[2] non fornisce alcuna soluzione.

Da questo esempio si può dunque dedurre che grazie all'approccio mixed-initiative ed alla possibilità di riconfigurare il piano, dopo una modifica dei *goals* che rende il piano inconsistente, è possibile ottenere un nuovo piano valido con uno sforzo relativamente piccolo. Con il contributo dell'uomo, che possiede una consapevolezza sul dominio superiore al sistema, è possibile modificare il piano in maniera mirata all'obiettivo appena aggiunto, ottenendo un piano consistente a livello simbolico evitando di dover effettuare una ri-pianificazione da zero. Poiché, come si è visto, il nuovo piano potrebbe però non essere valido in termini numerici, grazie alla funzionalità di riconfigurazione offerta dal sistema, l'operatore ha la possibilità di richiedere un adattamento delle modalità di esecuzione delle azioni ai nuovi vincoli.

Uno dei principali vantaggi di questo approccio è dunque la necessità di un tempo minore per la realizzazione del nuovo piano: una operazione di ri-pianificazione da zero, per un problema di complessità simile a quella dell'esempio, normalmente può richiedere ore (per l'esempio in esame, come mostrato sopra, 10800 secondi, ovvero 3 ore, non sono risultati sufficienti per trovare un piano valido), mentre con l'approccio applicato possono bastare pochi minuti (considerando anche i tempi delle operazioni effettuate dall'uomo). Inoltre un altro beneficio di questo tipo di pianificazione è dato dal fatto che, mentre una completa ri-pianificazione comporta la perdita del piano corrente, il quale potrebbe essere stato ottenuto secondo particolari esigenze e tenendo conto di aspetti o conoscenze non espresse al sistema, l'utilizzo della ri-configurazione, al contrario, permette di mantenere inalterata la struttura di alto livello del piano, andando ad intaccare esclusivamente le modalità di esecuzione delle azioni.

In situazioni di questo tipo può inoltre esser tenuta in considerazione un'ulteriore funzionalità messa a disposizione dal sistema. Se anche una ri-configurazione

[1] Tempi calcolati su macchina dotata di SO Linux Mint 12 64bit, Intel Core i3-2367M CPU @ 1.40GHz x 4, 4GB di RAM.

[2] Al pianificatore Metric-FF, per la ricerca di un piano è stato impostato un timeout di 10800 secondi, ovvero 3 ore, e la ricerca è terminata per scadenza del timeout.

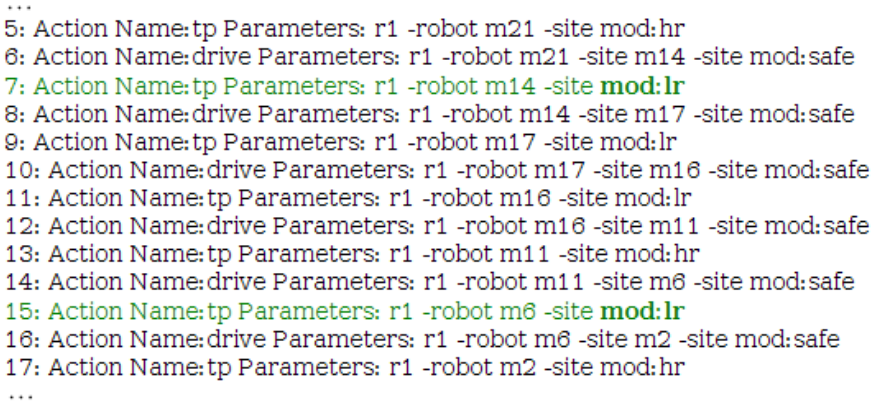


Figura 7.8: Risultato dell'operazione di configurazione delle due operazioni di `tp`

è ritenuta troppo invasiva in quanto può alterare le modalità di qualsiasi azione del piano, è possibile specificare quali azioni si intende configurare, impostando le loro modalità a *decided by system* e richiedendone una configurazione.

Tale operazione permette di effettuare una ricerca di una configurazione del piano che vada ad intaccare solamente le modalità non già specificate dall'utente (ovvero quelle indicate come *decided by system*), mantenendo inalterato il resto. Con questa operazione la complessità temporale viene ulteriormente ridotta , così come la perdita delle caratteristiche precedenti del piano.

Impostando a *decided by system* le due azioni di `tp` inserite nell'esempio precedente, il sistema è in grado di trovare (vedi figura 7.8), con l'operazione *Configure*, una soluzione perfettamente identica all'originale, semplicemente modificando le modalità di esecuzione delle nuove azioni.

7.2 Adattamento piano (Nuovi Vincoli su Risorse)

Il secondo scenario mostra quali possibilità sono messe a disposizione dell'utente per adattare un piano ad un problema in cui sono stati modificati i vincoli sulle risorse numeriche.

Supponendo di avere un piano valido per un problema che presenta dei vincoli non particolarmente stretti, l'utente può desiderare che tale piano sia più robusto,

```
( = (fuel-used) 0.0)
( = (time-spent) 0.0)
```

(a) Stato iniziale dei fluenti numerici variabili del problema

```
((fuel-used) <  1799.6 )
((time-spent) <  400 )
```

(b) I *goals* numerici del problema

Figura 7.9: Stato iniziale e *goals* del problema di esempio

e continui ad essere valido anche con vincoli più restrittivi.

La modifica dei *goals* numerici dà la possibilità all'utente di verificare quanto robusto sia il piano sviluppato.

Se dopo la modifica dei vincoli si ottiene un piano non più consistente, all'operatore vengono messe a disposizione due strade: la ri-pianificazione e la riconfigurazione. Nel primo caso egli può delegare al sistema la ricerca automatica di un piano che soddisfi i nuovi vincoli. Nel secondo caso, invece, può richiedere al sistema la ricerca di una diversa configurazione delle modalità di esecuzione delle azioni che rispetti tutti i vincoli.

Consideriamo un problema relativo al dominio DriverLog (vedi appendice A.3) i cui valori iniziali e i *goals* sono mostrati in figura 7.9. Una soluzione è data dal piano in figura 7.10. Tale piano viene eseguito con un utilizzo di `fuel` pari a 936 (ovvero alla fine della sua esecuzione, il fluente numerico `fuel-used` vale 936). Se vogliamo ottenere un piano che invece utilizzi meno di 935 unità di carburante, possiamo restringere il vincolo numerico sul fluente `fuel-used`. Naturalmente riducendo il carburante da utilizzare a un valore minore di quello attuale, il piano corrente non sarà più valido numericamente.

Per ottenere nuovamente un piano valido possiamo tentare un approccio classico richiedendo una ri-pianificazione oppure tentare una ri-configurazione delle modalità di esecuzione delle sue azioni.

I vantaggi della riconfigurazione si notano chiaramente sull'esempio in esame. Il tempo richiesto per tale operazione infatti risulta essere irrisorio rispetto a quello necessario ad una completa ri-pianificazione: circa 1 secondo contro 1333.08 secondi[3]. Inoltre effettuare una riconfigurazione permette di mantenere inaltera-

[3]Tempi calcolati su macchina dotata di SO Linux Mint 12 64bit, Intel Core i3-2367M CPU @ 1.40GHz x 4, 4GB di RAM. La parte significativa dell'output di Metric-FF riportava: 1333.08 seconds searching, evaluating 2050972 states, to a max depth of 142. Total time: 1333.08 seconds

```
(boardtruck d1 t1 m3 )
(drivetruck-fast t1 m3 m4 d1 )
(disembarktruck d1 t1 m4 )
(walk d1 m4 p1 )
(walk d1 p1 m0 )
(boardtruck d1 t0 m0 )
(drivetruck-fast t0 m0 m1 d1 )
(disembarktruck d1 t0 m1 )
```

Figura 7.10: Un piano completamente specificato e valido per il problema 7.9

ta la struttura generale del piano già costruito. L'ordine e il tipo di azioni non viene infatti modificato, le uniche modifiche che vengono effettuate riguardano le modalità di esecuzione delle azioni. Per l'esempio corrente una riconfigurazione comporta unicamente la modifica delle modalità della seconda e della settima azione a `normal`, come mostrato in figura 7.11

Analizzando scenari di questo tipo si può dunque notare come il sistema sviluppato fornisca all'utente molteplici funzionalità all'interno di un processo di pianificazione. L'approccio mixed-initiative in questo caso infatti, permette all'utente di manipolare a suo piacimento i *goals* del problema, per poi chiedere al sistema di verificare la validità del piano con le nuove condizioni. In base al risultato, l'operatore ha nuovamente la possibilità di decidere se agire manualmente, eventualmente modificando (come visto nello scenario 7.1) il piano corrente, oppure richiedere delle operazioni automatiche per ottenere un nuovo piano valido. In questo secondo caso, inoltre, il sistema mette a disposizione dell'utente ulteriori scelte (ri-pianificazione, riconfigurazione oppure configurazione solo di alcune modalità) che potranno essere fatte in base alle esigenze e alla conoscenza umana.

```
0: Action Name:boardtruck Parameters: d1 -driver t1 -truck m3 -location mod:default
1: Action Name:drivetruck Parameters: t1 -truck m3 -location m4 -location d1 -driver mod:normal
2: Action Name:disembarktruck Parameters: d1 -driver t1 -truck m4 -location mod:default
3: Action Name:walk Parameters: d1 -driver m4 -location p1 -location mod:default
4: Action Name:walk Parameters: d1 -driver p1 -location m0 -location mod:default
5: Action Name:boardtruck Parameters: d1 -driver t0 -truck m0 -location mod:default
6: Action Name:drivetruck Parameters: t0 -truck m0 -location m1 -location d1 -driver mod:normal
7: Action Name:disembarktruck Parameters: d1 -driver t0 -truck m1 -location mod:default
```

Figura 7.11: Il piano ottenuto con l'operazione di riconfigurazione

```
(board-normal p7 plane0 city0 )
(board-normal p3 plane0 city0 )
(fly-zoom plane0 city0 city3 )
(board-normal p4 plane0 city3 )
(board-normal p1 plane0 city3 )
(fly-zoom plane0 city3 city0 )
(debark-normal p4 plane0 city0 )
(fly-zoom plane0 city0 city1 )
(board-normal p0 plane0 city1 )
(board-normal p6 plane0 city1 )
(debark-normal p1 plane0 city1 )
(fly-cruise plane0 city1 city2 )
(board-normal p10 plane0 city2 )
(board-normal p5 plane0 city2 )
(debark-normal p7 plane0 city2 )
(debark-normal p3 plane0 city2 )
(debark-normal p6 plane0 city2 )
(fly-cruise plane0 city2 city3 )
(debark-normal p5 plane0 city3 )
(fly-zoom plane0 city3 city4 )
(debark-normal p10 plane0 city4 )
(debark-normal p0 plane0 city4 )
```

Figura 7.12: Un piano completamente specificato e valido per il problema dello scenario 7.3

7.3 Configurazione Assistita Piani Parzialmente Specificati

In quest'ultimo scenario viene mostrata l'utilità dei piani parzialmente specificati (*MMASkeletonPlan*) e della possibilità della loro configurazione offerta dal sistema.

L'esempio che mostriamo è tratto dal dominio ZenoTravel (vedi Appendice A.2) ma la stessa situazione è riproducibile anche su altri domini.

Supponendo di avere un piano completamente specificato come in figura 7.12, valido per un dato problema, l'operatore umano durante il processo di pianificazione può venire a conoscenza di (o dedurre) informazioni che vincolano alcune delle azioni del piano a determinate modalità di esecuzione. Oppure, in accordo con un

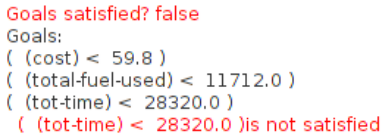


Figura 7.13: Output del controllo numerico sul piano con le prime tre modalità modificate

approccio mixed-initiative, l'operatore può voler testare il piano con configurazioni diverse delle modalità.

Naturalmente il sistema mette a disposizione operazioni per la scelta delle modalità delle azioni, tuttavia può facilmente accadere che dopo aver effettuato le modifiche il piano non sia più valido numericamente.

Se l'utente non necessita di richiedere una particolare modalità per tutte le azioni del piano (in tal caso se il piano non è valido numericamente occorre rivedere gli obiettivi del problema), è possibile rendere il piano corrente un piano parzialmente specificato impostando a *decided by system* tutte le modalità per cui non si hanno preferenze.

Il piano completamente specificato in figura 7.12, per esempio, se volessimo eseguire le prime tre azioni rispettivamente con le modalità `express, express` e `cruise`, non sarebbe numericamente consistente, come mostrato nella figura 7.13 che riporta l'output del check numerico sul piano modificato. Utilizzando invece un *MMASkeletonPlan* in cui tutte le modalità per le quali non vi sono particolari richieste sono impostate a *decided by system*, l'utente ha la possibilità di richiedere al sistema una configurazione esclusivamente di tali modalità (*Configure*). L'utente richiede cioè la ricerca di una soluzione costituita dalle azioni già presenti nel piano e che rispetti la scelta delle modalità già fissate.

In figura 7.14 è riportata la soluzione trovata per l'esempio corrente dall'operazione di configurazione.

Come si può notare, una funzionalità di questo tipo è particolarmente importante per una pianificazione interattiva. In questo caso infatti nè una ripianificazione, nè una riconfigurazione sarebbero state operazioni adeguate alle necessità dell'utente.

I piani parzialmente specificati presentano una serie di informazioni in meno

0: Action Name:board Parameters: p7 -person plane0 -aircraft city0 -city mod:express
1: Action Name:board Parameters: p3 -person plane0 -aircraft city0 -city mod:express
2: Action Name:fly Parameters: plane0 -aircraft city0 -city city3 -city mod:cruise
3: Action Name:board Parameters: p4 -person plane0 -aircraft city3 -city **mod:normal**
4: Action Name:board Parameters: p1 -person plane0 -aircraft city3 -city **mod:normal**
5: Action Name:fly Parameters: plane0 -aircraft city3 -city city0 -city **mod:zoom**
6: Action Name:debark Parameters: p4 -person plane0 -aircraft city0 -city **mod:normal**
7: Action Name:fly Parameters: plane0 -aircraft city0 -city city1 -city **mod:zoom**
8: Action Name:board Parameters: p0 -person plane0 -aircraft city1 -city **mod:normal**
9: Action Name:board Parameters: p6 -person plane0 -aircraft city1 -city **mod:normal**
10: Action Name:debark Parameters: p1 -person plane0 -aircraft city1 -city **mod:normal**
11: Action Name:fly Parameters: plane0 -aircraft city1 -city city2 -city **mod:zoom**
12: Action Name:board Parameters: p10 -person plane0 -aircraft city2 -city **mod:normal**
13: Action Name:board Parameters: p5 -person plane0 -aircraft city2 -city **mod:normal**
14: Action Name:debark Parameters: p7 -person plane0 -aircraft city2 -city **mod:normal**
15: Action Name:debark Parameters: p3 -person plane0 -aircraft city2 -city **mod:normal**
16: Action Name:debark Parameters: p6 -person plane0 -aircraft city2 -city **mod:normal**
17: Action Name:fly Parameters: plane0 -aircraft city2 -city city3 -city **mod:zoom**
18: Action Name:debark Parameters: p5 -person plane0 -aircraft city3 -city **mod:normal**
19: Action Name:fly Parameters: plane0 -aircraft city3 -city city4 -city **mod:zoom**
20: Action Name:debark Parameters: p10 -person plane0 -aircraft city4 -city **mod:normal**
21: Action Name:debark Parameters: p0 -person plane0 -aircraft city4 -city **mod:normal**

Figura 7.14: Il piano ottenuto con l'operazione di configurazione

rispetto ai piani normali (le modalità non specificate). Tuttavia, durante la configurazione, vengono trattati con una quantità superiore di vincoli (le modalità già specificate) rispetto ai piani completamente specificati. Ciò permette all'utente di esprimere facilmente al sistema nuove informazioni e nuovi vincoli che riguardano strettamente il piano e di costruire un piano in maniera completamente interattiva con la possibilità di decidere un gran numero di aspetti della soluzione finale.

Capitolo 8

Conclusioni

Nella tesi è stato sviluppato un sistema per la costruzione di piani attraverso un approccio mixed-initiative. L'obiettivo è stato quello di rendere più agevole e allo stesso tempo interattivo il processo di pianificazione. Come visto nel capitolo 4, un approccio classico alla pianificazione non è praticabile nella maggioranza dei problemi e dei domini riguardanti situazioni reali. La grande varietà di eventi possibili e la non determinatezza dell'ambiente rendono infatti un piano costruito manualmente (o completamente in modo automatico) non abbastanza robusto e versatile. Per queste ragioni si è deciso di adottare un approccio misto, attraverso il quale è possibile realizzare un piano più versatile e robusto grazie alla partecipazione dell'uomo nel processo di pianificazione. L'apporto dell'uomo risulta essere infatti particolarmente vantaggioso sia in termini di tempo, come mostrato nel capitolo 7, sia in termini di qualità del piano (vedi anche [4]).

A tal proposito si consideri un problema relativo al dominio dei rover (A.1) contenente tra i *goals* delle informazioni da inviare a terra e una condizione del tipo `(comm_cost < 2)`.

Gli effetti di una operazione di `comm` sul `comm_cost` sono
`(increase (comm_cost) 1)` nel caso di modalità `ch1` e
`(increase (comm_cost) 3)` nel caso di modalità `ch2`.

Inoltre nessun'altra azione ha effetti su tale fluente numerico.

Ad un operatore umano è evidente che, se esiste una soluzione per il problema dato, essa dovrà contenere una e una sola azione `comm` effettuata in modalità `ch1`.

In un pianificatore come Metric-FF, invece, tale conoscenza non è presente.

La ricerca di un piano come quello utilizzato nello scenario 7.1 per il problema 7.1, il quale presenta proprio una situazione analoga a quella appena descritta, richiede un tempo molto lungo[1] e dunque un uso delle risorse computazionali eccessivo.

La maggiore consapevolezza dell'uomo e la sua capacità di ragionare in maniera diversa dal pianificatore, vanno dunque a rendersi indispensabili in situazioni di questo tipo. Un approccio mixed-initiative, che permette all'uomo di mettere a disposizione le proprie capacità e allo stesso tempo sfrutta le potenzialità computazionali del calcolatore, risulta essere la scelta vincente per una migliore pianificazione.

Inoltre può capitare durante il processo di pianificazione che non si conoscano sin dall'inizio tutti gli obiettivi che si vogliono raggiungere oppure la perfetta situazione dello stato iniziale del mondo in cui si andrà ad operare.

Per queste ragioni l'ambiente realizzato per questa tesi adotta un approccio misto. A tal scopo, come presentato nel capitolo 5, sono state rese possibili all'utente operazioni di modifica sia del problema (e in particolare dei suoi *goals*), sia del piano (del quale è possibile alterare ogni azione, ogni parametro e ogni modalità). Il sistema è stato sviluppato per l'utilizzo di piani multi-modalità, i quali, come visto nel capitolo 3, permettono una maggiore versatilità nel caso di incontingenze a tempo di esecuzione e, come estensione di essi è stata resa possibile la realizzazione di piani parzialmente specificati che costituiscono, come chiarito nella sezione 6.2, la maggior rappresentazione dell'interazione tra l'utente e il calcolatore nella costruzione di un piano.

Oltre a queste funzionalità manuali, sono state messe a disposizione dell'utente per il processo di pianificazione operazioni automatiche di realizzazione di un piano, quali la pianificazione tramite il pianificatore Metric-FF, la riconfigurazione di un piano completamente specificato e la configurazione di un piano parzialmente specificato. Inoltre l'operatore ha anche la possibilità di richiedere controlli di consistenza sia proposizionale che numerica o di esistenza di una

[1] Dopo 7 ore (25200 secondi) di ricerca su una macchina dotata di SO Linux Mint 12 32bit, Intel Pentium(R) Dual CPU E2160 @ 1.80GHz x 2, 3GB RAM, il pianificatore Metric-FF ancora non è in grado di restituire una soluzione.

possibile configurazione per un piano parzialmente specificato.

Tutto ciò garantisce all'operatore un alto livello di autonomia e controllo durante la costruzione di un piano. Allo stesso tempo, però, mette a disposizione gli strumenti computazionali necessari per una pianificazione efficiente che sfrutti le capacità di *searching* e *reasoning* proprie del sistema e delle sue componenti.

8.1 Sviluppi futuri

Il sistema realizzato, per via dei ristretti tempi di sviluppo, è aperto a molti miglioramenti e ottimizzazioni. Una possibile estensione potrebbe riguardare l'inserimento delle azioni all'interno del piano. L'operatore potrebbe essere a conoscenza di azioni che intende far eseguire all'agente intelligente, ma non essere in grado di decidere dove inserirle all'interno del piano. In tal caso sarebbe vantaggiosa una funzionalità che permetta di specificare un insieme di azioni che si intendono inserire nel piano, per poi delegare al sistema la ricerca di una posizione di inserimento consistente. Questo costituirebbe un ulteriore fattore di interazione durante la costruzione del piano, abbracciando appieno l'approccio mixed-initiative.

Un'altra possibile estensione riguarda sicuramente gli aspetti di ottimizzazione del piano. Nella versione attuale del sistema, per questioni di tempo, non è stato affrontato il problema dell'ottimizzazione. Esso prevede l'inserimento di quella che viene chiamata *metrica* all'interno della definizione del problema. Una semplice metrica può richiedere, ad esempio, di minimizzare il tempo di esecuzione di un piano. Il sistema, in questo caso, dovrebbe cercare all'interno dello spazio delle soluzioni una combinazione di azioni che utilizzi il minimo tempo possibile, a discapito naturalmente di altre risorse quali, ad esempio, nel dominio dei rover, il carburante o la qualità delle informazioni catturate. Questa funzionalità si associa molto bene alla possibilità di richiedere al sistema una configurazione/riconfigurazione di un piano. Infatti definiti dei parametri di ottimizzazione è possibile richiedere al sistema di generare un piano (o di configurare/riconfigurare quello esistente) che li ottimizzi.

Appendice A

Domini MMA

In questo capitolo vengono forniti in modo completo i domini ai quali si è fatto riferimento durante la tesi. Per lo sviluppo del sistema il principale dominio su cui ci si è concentrati è il dominio dei Rover. Tuttavia è bene ricordare che il sistema è stato implementato in modo tale da essere indipendente dal dominio.

A.1 Dominio dei Rover

```
(define (domain planetary-rover)
  (:requirements :typing :fluents)
  (:types robot - object site - object)
  (:predicates
      (in ?r - robot ?l - site)
      (road ?l -site ?l1 - site)
      (info ?r - robot ?l - site)
      (infoSent ?r -robot ?l -site)
  )
 (:functions
      (distance ?l1 ?l2 -site)
      (roughness ?l1 ?l2 -site)
      (comm_cost)
      (time)
      (memory ?r -robot)
```

```
    (memoryC ?r -robot)
    (power ?r -robot)
    (powerC ?r -robot)
    (infoLoss)
    (safe_cons ?r -robot)
    (safe_speed ?r -robot)
    (cruise_cons ?r -robot)
    (cruise_speed ?r -robot)
    (agile_cons ?r -robot)
    (agile_speed ?r -robot)
    (bandwidth-ch1 ?r -robot)
    (bandwidth-ch2 ?r -robot)
    (ch1-cons ?r -robot)
    (ch2-cons ?r -robot)
)
(:action drive
    :parameters ( ?r - robot ?l1 - site ?l2 - site)
    :modalities (safe,normal,agile)
    :precondition (and (in ?r ?l1) (road ?l1 ?l2)
        (safe:
          (>= (power ?r)
              (* (safe_cons ?r)
                 (/ (distance ?l1 ?l2) (safe_speed ?r)))))
        (cruise:
          (>= (power ?r)
              (* (cruise_cons ?r)
                 (/ (distance ?l1 ?l2) (cruise_speed ?r)))))
        (agile:
          (>= (power ?r)
              (* (agile_cons ?r)
                 (/ (distance ?l1 ?l2) (agile_speed ?r))))))
    :effect (and (in ?r ?l2) (not (in ?r ?l1))
        (safe:
          (decrease (power ?r)
                    (* (safe_cons ?r)
```

```
                    (/ (distance ?l1 ?l2) (safe_speed ?r))))
        (increase (time)
                  (/ (distance ?l1 ?l2)) (safe_speed ?r)))
        (increase (powerC ?r)
                  (* (safe_cons ?r)
                    (/ (distance ?l1 ?l2) (safe_speed ?r))))
      (cruise:
        (decrease (power ?r)
               (* (cruise_cons ?r)
                  (/ (distance ?l1 ?l2) (cruise_speed ?r))))
        (increase (time)
                  (/ (distance ?l1 ?l2)) (cruise_speed ?r))
        (increase (powerC ?r)
               (* (cruise_cons ?r)
                  (/ (distance ?l1 ?l2) (cruise_speed ?r)))))
      (agile:
        (decrease (power ?r)
               (* (agile_cons ?r)
                  (/ (distance ?l1 ?l2) (agile_speed ?r))))
        (increase (time)
                  (/ (distance ?l1 ?l2)) (agile_speed ?r))
        (increase (powerC ?r)
               (* (agile_cons ?r)
                  (/ (distance ?l1 ?l2) (agile_speed ?r))))))
)
(:action tp
    :parameters ( ?r - robot ?l1 - site )
    :modalities (lr,hr)
    :precondition (and(in ?r ?l1)
          (lr: (>= (memory ?r) 1))
          (hr: (>= (memory ?r) 2)))
    :effect (and (info ?r ?l1)
          (lr:
            (increase (memoryC ?r) 1)
            (decrease (memory ?r) 1)
```

```
            (increase (time) 1)
            (increase (infoLoss) 3))
          (hr:
            (increase (memoryC ?r) 2)
            (decrease (memory ?r) 2)
            (increase (time) 1)
            (increase (infoLoss) 1)))
)
(:action comm
    :parameters ( ?r - robot ?l1 - site )
    :modalities (ch1,ch2)
    :precondition (and(in ?r ?l1)
          (ch1:
            (and (> (memoryC ?r) 0)
                 ( >= (power ?r)
                      (/ (memoryC ?r) (bandwith-ch1 ?r)))))
          (ch2:
            (and (> (memoryC ?r) 0)
                 ( >= (power ?r)
                      (/ (memoryC ?r) (bandwith-ch2 ?r))))))
    :effect (and (infoSent ?r ?l1)
          (ch1:
            (assign (memoryC ?r) 0)
            (assign (memory ?r) (memoryC ?r))
            (increase (time)
                      (/ (memoryC ?r) (bandwith-ch1 ?r)))
            (increase (powerC ?r)
                      (* (ch1-cons ?r)
                         (/ (memoryC ?r) (bandwith-ch1 ?r)))
            (decrease (power ?r)
                      (* (ch1-cons ?r)
                         (/ (memoryC ?r) (bandwith-ch1 ?r)))
            (increase (comm_cost) 1))))
          (ch2:
            (assign (memoryC ?r) 0)
```

```
              (assign (memory ?r) (memoryC ?r))
              (increase (time)
                        (/ (memoryC ?r) (bandwith-ch2 ?r)))
              (increase (powerC ?r)
                        (* (ch2-cons ?r)
                           (/ (memoryC ?r) (bandwith-ch2 ?r))))
              (decrease (power ?r)
                        (* (ch2-cons ?r)
                           (/ (memoryC ?r) (bandwith-ch2 ?r))))
              (increase (comm_cost) 3)))
  )
)
```

A.2 Dominio ZenoTravel

```
(define (domain zeno-travel)
  (:requirements :typing :fluents)
  (:types
      locatable city - object
      aircraft person -locatable)
  (:predicates
      (located ?x - locatable ?c - city)
      (in ?p - person ?a - aircraft))
  (:functions
      (fuel ?a - aircraft)
      (distance ?c1 - city ?c2 - city)
      (cruise-burn ?a - aircraft)
      (fast-burn ?a - aircraft)
      (inv-zoom-speed ?a - aircraft)
      (inv-cruise-speed ?a - aircraft)
      (capacity ?a - aircraft)
      (total-fuel-used)
      (onboard ?a - aircraft)
      (zoom-limit ?a - aircraft)
      (tot-time)
```

```
    (cost)
)
(:action board
    :parameters (?p - person ?a - aircraft ?c - city)
    :modalities (normal,express)
    :precondition (and (located ?p ?c) (located ?a ?c))
    :effect (and (not (located ?p ?c)) (in ?p ?a)
          (normal:
            (increase (onboard ?a) 1)
            (increase (tot-time) 240)
            (increase (cost) 1))))
          (express:
            (increase (onboard ?a) 1)
            (increase (tot-time) 120)
            (increase (cost) 4)))
)
(:action debark
    :parameters (?p - person ?a - aircraft ?c - city)
    :modalities (normal,express)
    :precondition (and (in ?p ?a)(located ?a ?c))
    :effect (and (not (in ?p ?a))(located ?p ?c)
          (normal:
            (decrease (onboard ?a) 1)
            (increase (tot-time) 240)
            (increase (cost) 1)))
          (express:
            (decrease (onboard ?a) 1)
            (increase (tot-time) 120)
            (increase (cost) 1)))
)
(:action fly
    :parameters (?a - aircraft ?c1 ?c2 - city)
    :modalities (cruise,zoom)
    :precondition (and (located ?a ?c1)
          (cruise: (>= (fuel ?a)
```

```
                        (* (distance ?c1 ?c2) (cruise-burn ?a))))
            (zoom: (>= (fuel ?a)
                       (* (distance ?c1 ?c2) (zoom-burn ?a))))
      :effect (and (not (located ?a ?c1))(located ?a ?c2)
            (cruise:
              (increase (total-fuel-used)
                        (* (distance ?c1 ?c2) (cruise-burn ?a)))
              (decrease (fuel ?a)
                        (* (distance ?c1 ?c2) (cruise-burn ?a)))
              (increase (tot-time)
                     (* (distance ?c1 ?c2) (inv-cruise-speed ?a)))
            (zoom:
              (increase (total-fuel-used)
                        (* (distance ?c1 ?c2) (zoom-burn ?a)))
              (decrease (fuel ?a)
                        (* (distance ?c1 ?c2) (zoom-burn ?a)))
              (increase (tot-time)
                     (* (distance ?c1 ?c2) (inv-zoom-speed ?a)))
  )
  (:action refuel
      :parameters (?a - aircraft ?c - city)
      :precondition (and (> (capacity ?a) (fuel ?a))
                         (located ?a ?c))
      :effect (and (assign (fuel ?a) (capacity ?a))
                   (increase (tot-time) 900)))
  )
)
```

A.3 Dominio DriverLog

```
(define (domain driverlog)
  (:requirements :typing :fluents)
  (:types
      location locatable - object
      driver truck obj - locatable)
```

```
(:predicates
    (pos ?obj - locatable ?loc - location)
    (in ?obj1 - obj ?obj - truck)
    (driving ?d - driver ?v - truck)
    (link ?x ?y - location)
    (path ?x ?y - location)
    (empty ?v - truck))
(:functions
    (time-to-walk ?l1 ?l2 - location)
    (time-to-drive-fast ?l1 ?l2 - location)
    (time-to-drive-normal ?l1 ?l2 - location)
    (fuel-used)
    (time-spent)
    (fuel-per-minute-fast ?t - truck)
    (fuel-per-minute-normal ?t - truck)
    (load ?t - truck)
    (capacity ?t - truck))
(:action loadtruck-safe
    :parameters(
       ?obj - obj
       ?truck - truck
       ?loc - location)
    :precondition (and
       (pos ?truck ?loc) (pos ?obj ?loc) (empty ?truck)
       (< (load ?truck) (capacity ?truck)) )
    :effect (and
       (not (pos ?obj ?loc))
       (in ?obj ?truck)
       (increase (load ?truck) 1)
       (increase (fuel-per-minute-fast ?truck) (+ (load ?truck) 1))
       (increase (fuel-per-minute-normal ?truck) (+ (load ?truck) 1))
       (increase (time-spent) 4)))

(:action loadtruck-normal
    :parameters (
```

```
        ?obj - obj
        ?truck - truck
        ?loc - location)
    :precondition (and
        (pos ?truck ?loc) (pos ?obj ?loc) (empty ?truck)
        (< (load ?truck) (/ (capacity ?truck) 2)))
    :effect (and
        (not (pos ?obj ?loc))
        (in ?obj ?truck)
        (increase (load ?truck) 1)
        (increase (fuel-per-minute-fast ?truck) (+ (load ?truck) 1))
        (increase (fuel-per-minute-normal ?truck) (+ (load ?truck) 1))
        (increase (time-spent) 2)))

(:action unloadtruck
    :parameters (
        ?obj - obj
        ?truck - truck
        ?loc - location)
    :precondition(and
        (pos ?truck ?loc) (in ?obj ?truck) (empty ?truck))
    :effect(and
        (not (in ?obj ?truck))
        (pos ?obj ?loc)
        (decrease (load ?truck) 1)
        (decrease (fuel-per-minute-fast ?truck) (load ?truck))
        (decrease (fuel-per-minute-normal ?truck) (load ?truck))
        (increase (time-spent) 2)))

(:action boardtruck
    :parameters (
        ?driver - driver
        ?truck - truck
        ?loc - location)
    :precondition (and
```

```
        (pos ?truck ?loc) (pos ?driver ?loc) (empty ?truck))
    :effect (and
        (not (pos ?driver ?loc))
        (driving ?driver ?truck)
        (not (empty ?truck))
        (increase (time-spent) 1)))

(:action disembarktruck
    :parameters (
        ?driver - driver
        ?truck - truck
        ?loc - location)
    :precondition
        (and (pos ?truck ?loc) (driving ?driver ?truck))
    :effect (and
        (not (driving ?driver ?truck))
        (pos ?driver ?loc)
        (empty ?truck)
        (increase (time-spent) 1)))

(:action drivetruck-fast
    :parameters (
        ?truck - truck
        ?loc-from - location
        ?loc-to - location
        ?driver - driver)
    :precondition (and
        (pos ?truck ?loc-from)
        (driving ?driver ?truck)
        (link ?loc-from ?loc-to))
    :effect (and
        (not (pos ?truck ?loc-from))
        (pos ?truck ?loc-to)
        (increase (fuel-used)
            (* (fuel-per-minute-fast ?truck)
```

```
                (time-to-drive-fast ?loc-from ?loc-to)))
        (increase (time-spent)
            (time-to-drive-fast ?loc-from ?loc-to))
))


  (:action drivetruck-normal
      :parameters (
         ?truck - truck
         ?loc-from - location
         ?loc-to - location
         ?driver - driver)
      :precondition (and
         (pos ?truck ?loc-from)
         (driving ?driver ?truck)
         (link ?loc-from ?loc-to))
      :effect (and
         (not (pos ?truck ?loc-from))
         (pos ?truck ?loc-to)
         (increase (fuel-used)
             (* (fuel-per-minute-normal ?truck)
                (time-to-drive-normal ?loc-from ?loc-to)))
         (increase (time-spent)
             (time-to-drive-normal ?loc-from ?loc-to))))

  (:action walk
      :parameters(
         ?driver - driver
         ?loc-from - location
         ?loc-to - location)
      :precondition(and
         (pos ?driver ?loc-from)
         (path ?loc-from ?loc-to))
      :effect (and
         (not (pos ?driver ?loc-from))
```

```
        (pos ?driver ?loc-to)
        (increase (time-spent)
            (time-to-walk ?loc-from ?loc-to))))
)
```

Bibliografia

[1] AIPS-98 Planning Competition Committee (1998): *PDDL - The Planning Domain Definition Language Version 1.2*.

[2] John L. Bresina & Paul H. Morris (2007): *Mixed-Initiative Planning in Space Mission Operations*. *AI Magazine* Volume 28(Number 2), pp. 75–88.

[3] Michael T. Cox & Chen Zhang (2007): *Mixed-Initiative Goal Manipulation*. *AI Magazine* Volume 28(Number 2), pp. 62–73.

[4] M.L. Cummings, Jonathan P. How, Andrew Whitten & Olivier Toupet (2012): *The Impact of Human-Automation Collaboration in Decentralized Multiple Unmanned Vehicle Control*. *Proceedings Of The IEEE* Vol. 100(No. 3), pp. 660–671.

[5] M. Fox & D. Long (2003): *PDDL 2.1 : An Extension to pddl for Expressing Temporal Planning Domains*. *Journal of Artificial Intelligence Research* 20, pp. 61–124.

[6] Jörg Hoffmann (2001): *FF: The Fast-Forward Planning System*. *AI Magazine* 22(3), pp. 57–62.

[7] Jörg Hoffmann (2003): *The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables*. *Journal of Artificial Intelligence Research* (20), pp. 293–341.

[8] Stuart J. Russell & Peter Norvig (2009): *Artificial Intelligence: A Modern Approach*, 3rd edition. Prentice Hall series in artificial intelligence.

[9] Enrico Scala (2013): *Reconfiguration and Replanning for Robust Execution of Plans Involving Continuous and Consumable Resources.* Ph.D. thesis, Università degli Studi di Torino.

[10] Enrico Scala, Roberto Micalizio & Pietro Torasso (2013): *Adapting Planetary Rover Plans via Action Modality Reconfiguration.* In Luis Castillo Vidal, Steve Chien & Riccardo Rasconi, editors: *Proceedings of the 7th Scheduling and Planning Applications woRKshop*, pp. 41–48.