UNIVERSITÀ DEGLI STUDI DI TORINO
Facoltà di Scienze Matematiche Fisiche e Naturali
Corso di Laurea Magistrale in Informatica

Progetto di Laboratorio

NG Café

Davide Dell'Anna, Matteo Madeddu, Enrico Mensa

Indice

Capitolo 1

Il sistema

1.1 Introduzione

L'agente che abbiamo sviluppato è un agente single-minded.

Il sistema è quindi in grado di portare a termine solo un goal alla volta ma, in funzione delle sue percezioni può decidere di abortirlo qualora lo ritenga irraggiungibile. Le strutture dati che vengono mantenute all'interno dell'agente sono però state sviluppate con un orientamento open-minded: i goal ad alto livello sono infatti suddivisi in diversi atomi (detti *task*) che vengono pianificati singolarmente e che quindi sono predisposti ad una eventuale futura implementazione in cui l'agente può decidere di riconsiderare le sue intenzioni ad ogni ciclo di esecuzione.

1.2 Architettura dell'agente

L'architettura che abbiamo sviluppato prende ispirazione dal modello del *procedural reasoning system* e coinvolge diversi moduli. Ne vediamo uno schema riassuntivo in Figura 1.1.

1.2.1 Descrizione dei moduli

Forniremo prima di tutto una breve descrizione del funzionamento di ogni modulo per poi mostrarne l'interazione in fase di esecuzione dell'agente.

Modulo `AGENT`**.** Il modulo `AGENT` implementa una sorta di router che coordina l'interazione fra le varie sezioni dell'agente ma che di per sé non effettua alcuna operazione. Starà quindi a questo modulo capire quando interpretare i messaggi dell'`ENV` o le percezioni, quando entrare in deliberation e quando andare ad eseguire la successiva azione del piano corrente.

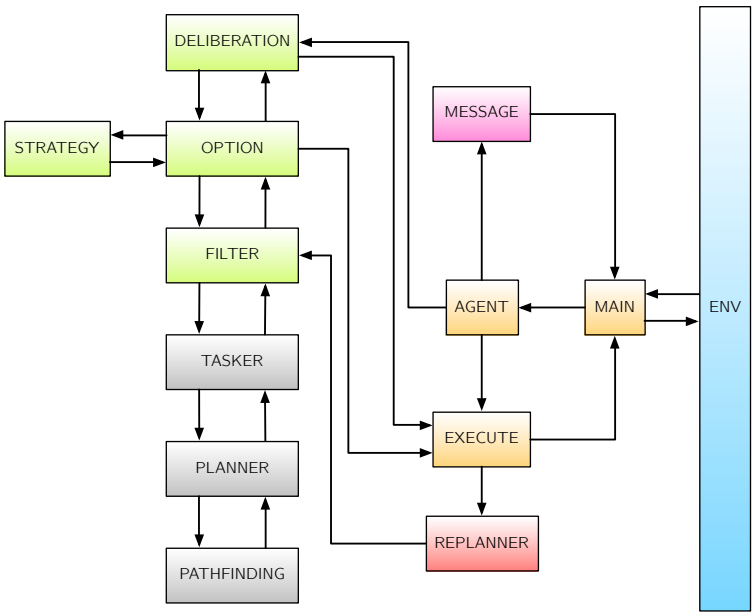

Figura 1.1: Schema generale dei moduli.

`AGENT` si occupa inoltre, in fase di lancio dell'agente, dell'istanziazione dei fatti relativi all'ambiente, come ad esempio i fatti `K-cell` e `service-cell`[1].

Modulo `SERVICE-CELL`. Il modulo `SERVICE-CELL` è un modulo che viene utilizzato una sola volta in fase di inizializzazione dell'agente e si occupa di trovare le service cell di ogni oggetto e di produrre per ciascuna di esse un fatto `service-cell`. Questa operazione è necessaria poiché per poter interagire con un certo oggetto l'agente dovrà trovarsi in una delle sue service cell e quindi i fatti prodotti da questo modulo saranno di fondamentale importanza in fase di ricerca del percorso da compiere.

Modulo `MESSAGE`. Il modulo `MESSAGE` è molto semplice e viene interpellato ad ogni ciclo di esecuzione. Scopo di questo modulo è l'elaborazione dei messaggi che l'agente riceve dall'ambiente e la loro trasformazione in goal di alto livello (fatti di tipo `goal`, Listing 1.1).

```
; Goal di alto livello
(deftemplate goal
    (slot id)
    (slot request-time)
    (slot step)
```

[1] Una service cell è una cella libera adiacente ad un oggetto tramite la quale l'oggetto stesso è accessibile.

```
    (slot sender)
    ; Slot popolati solo in caso di order
    (slot drink-order (default 0)) (slot food-order (default
        0))
    (slot food-to-deliver (default 0)) (slot drink-to-deliver
        (default 0))  ; traccia quanti cibi o bevande dobbiamo
        ancora consegnare

    ; Slot di struttura
    (slot type (allowed-values order clean trash))
    (slot status (allowed-values undone to-plan
        evaluating-best planned executing done aborted))
)
```

Listing 1.1: Template del fatto `goal`.

Nel caso in cui il messaggio ricevuto sia di ordine, dopo averlo interpretato, questo modulo produce direttamente una `exec` ed effettua un focus sul modulo `MAIN` al fine di compiere un'azione di `Inform`.

Modulo `PERCEPTION`**.** Il modulo `PERCEPTION` viene richiamato ad ogni ciclo di esecuzione ed il suo ruolo è quello di ricevere ed interpretare le percezioni provenienti dal mondo. In base alle percezioni ricevute viene aggiornato il contenuto delle `K-cell` e vengono ripristinate allo stato iniziale quelle `K-cell` che non sono più nel raggio visivo dell'agente.

Modulo `DELIBERATION`**.** Il modulo `DELIBERATION` è un modulo di servizio che prepara l'agente all'importante fase di scelta del goal da portare a termine. Si entra quindi in questo modulo soltanto se non vi sono piani correntemente in esecuzione (a causa della natura single-minded dell'agente).

L'operazione di deliberation consta di due fasi:

1. Si determinano i goal attuabili tra cui scegliere (modulo `OPTION`).
2. Fra le opzioni precedentemente ottenute se ne sceglie una in base alla strategia adottata (moduli `FILTER` e `STRATEGY`).

Una volta scelto un goal questo verrà pianificato e poi se ne inizierà l'esecuzione.

Moduli `OPTION` **e** `CHECK-OPTION`**.** Il modulo `OPTION` coordina l'interazione fra il modulo `STRATEGY` ed il modulo `CHECK-OPTION`.

In seguito alla scelta del goal più appetibile da parte del modulo `STRATEGY`, il

modulo CHECK-OPTION ne testa l'applicabilità. L'applicabilità di un goal dipende dallo stato interno dell'agente: ad esempio non è possibile intraprendere un goal di tipo ordine nel caso in cui l'agente sia sporco. Se il goal scelto dal modulo STRATEGY non dovesse essere applicabile, starà a quest'ultimo proporne un altro al modulo CHECK-OPTION.

Qualora nessun goal venga ritenuto applicabile, l'agente andrà direttamente al modulo EXECUTE dove verrà designata un'azione di default da eseguire.

Moduli STRATEGY ed EURISTIC. I moduli STRATEGY ed EURISTIC implementano la vera e propria capacità decisionale dell'agente. Compito del modulo EURISTIC è quello di prevedere le future azioni dell'agente nel caso in cui un determinato goal venga scelto e quindi prevedere quanta penalità verrà attribuita durante l'esecuzione del goal stesso. Sta poi al modulo STRATEGY combinare le informazioni fornite da EURISTIC in maniera appropriata e quindi scegliere un goal piuttosto che un altro.

Alcune delle strategie implementate non fanno uso di un'euristica, pertanto alcune di esse non sfruttano i dati prodotti dal modulo EURISTIC.

Modulo FILTER. Il modulo FILTER si occupa di generare i vari piani possibili per un goal scelto e valutare quale sia il migliore. Un goal è identificato da una serie di compiti atomici che chiamiamo *task*. Ad esempio, un ordine del tipo ''1 food e 0 drink'' può essere tradotto in due task: `taskLoadFood` e `taskDeliverOrder`. I due task, rispettivamente, identificano le azioni per raggiungere il `FoodDispenser` e caricare il cibo e quelle per raggiungere il tavolo e consegnare il cibo. L'adempimento di un task (e quindi del goal di cui fa parte) può avvenire facendo uso di diverse service cell: una configurazione di service cell identifica quindi una precisa versione di piano per un determinato goal. Rappresentiamo una versione di piano tramite il fatto `plan` (riportato in Listing 1.2).

```
(deftemplate plan
    (slot goal-id)
    (slot version)
    (slot cost (default 0))

    ; Service cell scelte
    (slot sc1)
    (slot sc2)
    (slot sc3)
    (slot sc4)
    (slot sc5)
```

```
    (slot status (allowed-values to-task tasked planning
        planned))
    (slot aborted (allowed-values yes no) (default no))
)
```

Listing 1.2: Template del fatto `plan`.

Dopo aver generato tutte le possibili versioni di piano per un goal (grazie al modulo `TASKER`), il modulo `FILTER` ne computa il costo (slot `cost`) e sceglie quello con costo minore, rimuovendo tutti gli altri.

Modulo `TASKER`. Per ogni versione di piano generata dal modulo `FILTER` viene richiamato il modulo `TASKER` che ne indica i task necessari per la sua realizzazione, sfruttando le service cell proprie della versione di piano. I task sono rappresentati mediante l'uso di fatti `task` (Listing 1.3).

```
(deftemplate task
    (slot goal-id)
    (slot plan-version)
    (slot id)
    (slot type (allowed-values taskLoadFood taskLoadDrink
        taskDeliverOrder taskTrashRB taskTrashTB
        taskCleanTable))
    (slot qt-food (default 0)) ;utile solo per taskLoadFood
        taskLoadDrink taskDeliverOrder
    (slot qt-drink (default 0)) ;utile solo per taskLoadFood
        taskLoadDrink taskDeliverOrder
    (slot target-id) ;identificare l'id dell'oggetto target
        per le planned action
    (slot service-cell-id) ;mantiene l'identificatore della
        cella service designata per il raggiungimento di
        target-id

    ; mantiene la cella di partenza
    (slot source-r)
    (slot source-c)
    (slot source-dir)
)
```

Listing 1.3: Template del fatto `task`.

Ogni fatto `task` è contraddistinto dalla sua tipologia, dal piano a cui fa riferimento e dal goal a cui appartiene. La serie di azioni volte al compimento di un task si può sempre dividere in due insiemi: il primo (eventualmente vuoto) contiene azioni di movimento utili a raggiungere uno specifico target mentre il secondo contiene le azioni necessarie ad interagire con esso.

Moduli `PLANNER` **e** `PATHFINDING`**.** Il modulo `PLANNER` elabora ogni fatto di tipo `task` corrispondente ad un piano attualmente in computazione e restituisce una sequenza di `planned-action` utili ad eseguire il task stesso. Una `planned-action` (Listing 1.4) è sostanzialmente una versione interna dell'`exec` che il modulo `ENV` è in grado di interpretare.

```
(deftemplate planned-action
    (slot goal-id)
    (slot plan-version)
    (slot task-id)
    (slot action-id)
    (slot oper)
    (slot param1 (default NA))
    (slot param2 (default NA))

    ; Manteniamo la situazione dell'agente "simulato" in
       pianificazione
    (slot pl-r)
    (slot pl-c)
    (slot pl-direction)
)
```

Listing 1.4: Template del fatto `planned-action`.

Alcune delle `planned-action` definite sono ottenute mediante l'uso del modulo `PATHFINDING` che implementa la strategia di ricerca in ampiezza informata $A^\star$.

A questo punto della computazione siamo giunti, dalla definizione di un goal ad alto livello, alla generazione delle azioni atomiche necessarie al suo compimento.

Modulo `EXECUTE`**.** Il modulo `EXECUTE` esegue una `planned-action` trasformandola in un fatto `exec`. È in questo modulo che vengono verificate le condizioni di applicabilità delle azioni: un'azione pianificata non è necessariamente applicabile. Pensiamo al caso in cui l'agente è ostacolato da una persona: in fase di pianificazione il percorso calcolato risultava essere libero, ma a seguito dell'evoluzione del mondo questo non è più vero. È quindi il modulo `EXECUTE` a richiamare il modulo `REPLANNER` al fine di generare un piano alternativo (qualora questo esista) per l'adempimento del goal attualmente in esecuzione.

Nel caso in cui non vi sia alcun goal in esecuzione l'agente compie un'azione di `Wait`. Questo non è sempre vero: ne parleremo in dettaglio nella Sezione 1.3.1.

Modulo `REPLANNER`**.** Il modulo `REPLANNER` imposta la ripianificazione in seguito al fallimento dell'esecuzione di un'azione pianificata. Le operazioni com-

piute da tale modulo constano nella retract di tutti i fatti relativi al goal attualmente in esecuzione e nel passaggio al modulo FILTER al fine di trovare un piano alternativo per il goal corrente a partire dallo stato attuale dell'agente.

1.2.2 Interazione tra i moduli

Per illustrare l'interazione tra i moduli abbiamo schematizzato i cinque scenari possibili di flusso.

Abbiamo distinto il flusso in due tipologie: andata e ritorno. Le linee blu rappresentano il flusso di chiamate che partono dall'ENV e vanno verso quei moduli che adempiono all'esigenza corrente, mentre le linee rosse identificano le ''chiamate di ritorno'' che riportano l'esecuzione all'ENV. Con le linee tratteggiate indichiamo invece quei collegamenti fra i moduli che non vengono sfruttati nel particolare flusso indicato.

Abbiamo rappresentato il flusso tra i moduli in caso di:

- Nuovo messaggio dall'ENV (Figura 1.2): il flusso di andata proveniente dall'ENV porta direttamente al modulo MESSAGE, nel quale viene prodotta un'azione di Inform inviata poi all'ENV, seguendo il flusso rosso. Questo schema rappresenta la situazione in cui l'ENV invia all'agente un messaggio di ordine, al quale deve seguire una risposta di accettazione o rifiuto.

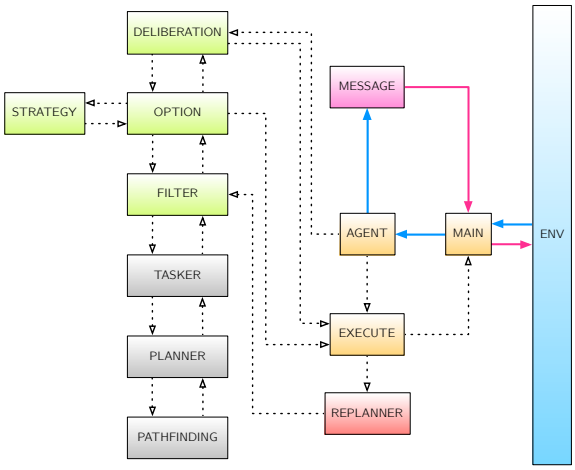

Figura 1.2: Flusso fra i moduli in caso di nuovo messaggio dall'ENV.

- Presa in carico di un nuovo goal (Figura 1.3): il flusso di andata ci porta direttamente al modulo DELIBERATION dove, tramite i moduli OPTION e STRATEGY si sceglie quale goal portare in esecuzione. Il modulo FILTER

genera tutti i piani possibili per quel goal facendo uso della pila `TASKER-PLANNER-PATHFINDING`. Come indicato dal flusso rosso, una volta conclusa la pianificazione si entra direttamente nel modulo `EXECUTE` per eseguire la prima delle `planned-action` appena generate.

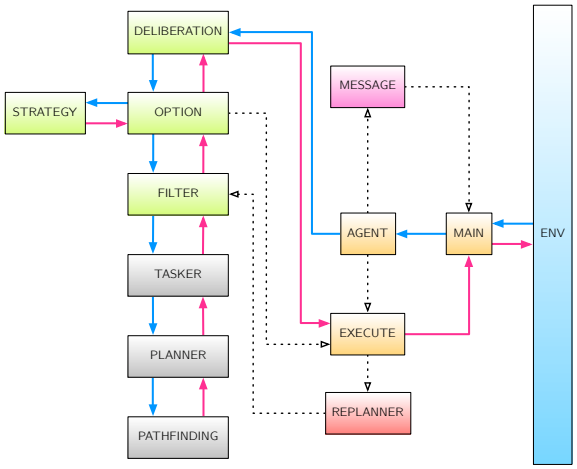

Figura 1.3: Flusso fra i moduli in caso di presa in carico di un nuovo goal.

- Esecuzione di un'azione precedentemente pianificata (Figura 1.4): il flusso di andata porta al modulo `EXECUTE` dove viene individuata la `planned-action` da eseguire che verrà trasformata in `exec` ed interpretata dal modulo `ENV`.

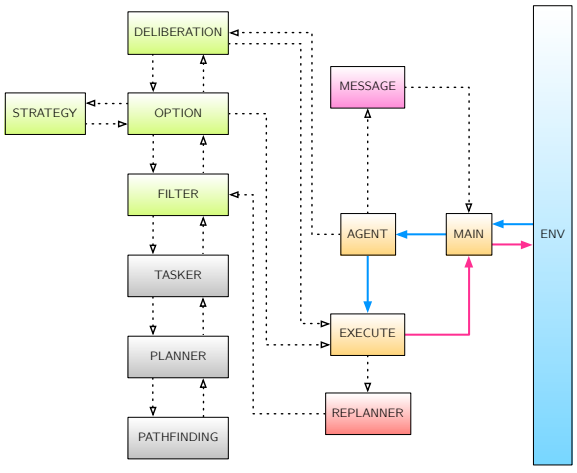

Figura 1.4: Flusso fra i moduli nel caso in cui si esegua un'azione precedentemente pianificata.

- Ripianificazione (Figura 1.5): qualora si debba ripianificare sarà il modulo `EXECUTE` a rendersene conto. Seguendo il flusso blu vediamo come il modulo `REPLANNER` richiami il modulo `FILTER` al fine di ottenere un piano alternativo per il goal appena abortito. La linea rossa illustra come il modulo `DELIBERATION` riporti il focus su `EXECUTE`. Tramite questo flusso ripercorriamo la strada di ritorno già illustrata in Figura 1.3; inoltre, nel caso in cui non esistano piani alternativi per portare a termine il goal è compito del modulo `DELIBERATION` selezionarne un altro.

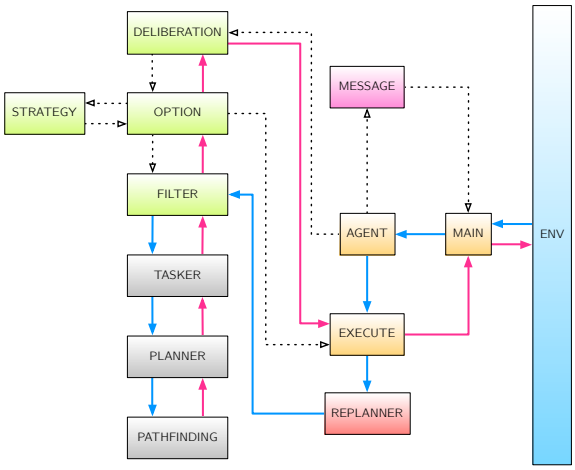

Figura 1.5: Flusso fra i moduli in situazione di ripianificazione.

- Nessun goal applicabile (Figura 1.6): qualora nessuno dei goal pendenti sia applicabile è compito del modulo `OPTION` richiamare il modulo `EXECUTE` al fine di eseguire un'azione di default.

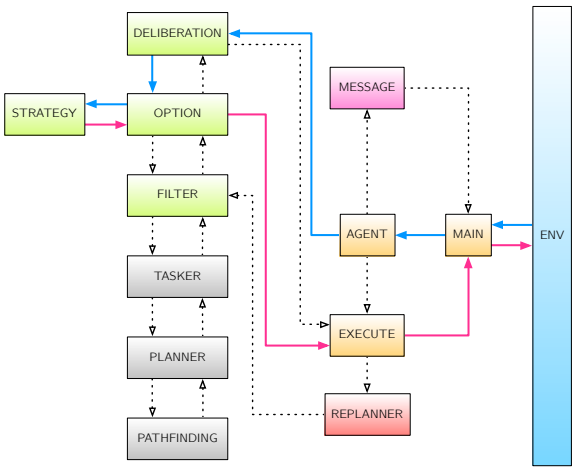

Figura 1.6: Flusso fra i moduli in caso non vi sia alcun goal applicabile.

1.3 Scelte implementative

Riportiamo in quest'ultima sezione alcune particolari features che abbiamo implementato indicando quali scelte implementative sono state compiute e perché.

1.3.1 Closeness detection

La closness detection è una feature che abbiamo implementato per impedire che il robot occupi una cella dove un agente umano desideri transitare. Questa situazione è molto particolare e si può verificare in due casi:

- Tutti i goal pendenti sono abortiti poiché irragiunbili e l'agente si trova vicino ad un agente umano.
- L'agente non ha alcun goal pendente e si trova vicino ad un agente umano.

In entrambi i casi si potrebbe attuare una azione di `Wait` rischiando però che il nostro agente ostruisca permanentemente il transito dell'agente umano.

La closness detection risolve questa situazione attivandosi in alternativa ad una `Wait` nel caso in cui una persona si trovi nel raggio visivo dell'agente. Il nostro agente cercherà quindi la prima cella libera intorno a sé (cercando di muoversi nel minor numero di azioni possibile) e vi si muoverà. Nel caso in cui un nuovo goal sopraggiunga oppure uno dei pendenti risulti nuovamente attuabile la feature di closeness detection verrà automaticamente disattivata a favore dell'esecuzione della prima `planned-action` del nuovo piano appena trovato.

Qualora un agente umano blocchi il nostro agente senza lasciargli via di fuga preferiamo compiere una azione di `Wait` piuttosto che effettuare una serie di `Turn`: la ragione di questa scelta è puramente estetica, dal punto di vista delle penalità sarebbe preferibile lasciar girare l'agente su se stesso (discuteremo di questo fatto nel Capitolo 4).

1.3.2 Pianificazione intelligente

Come già descritto in precedenza, per ogni goal vengono generate molte versioni di piano che utilizzano tutte le possibili permutazioni di service cell. È quindi evidente che la costruzione delle varie versioni comporti la ripetizione della pianificazione di task identici. Onde evitare inutile computazione, quando il modulo `PLANNER` deve inizializzare la pianificazione per un determinato task verifica se in un'altra versione di piano per lo stesso goal, quel task è già stato pianificato. In tal caso il `PLANNER` si limiterà a copiare le `planned-action` già

calcolate anziché chiedere al `PATHFINDING` di generarne di nuove.

Questa feature apre le porte ad una grande miglioria: con alcune lievi modifiche sarebbe molto semplice permettere la copia di `planned-action` non solo da altre versioni di piano per lo stesso goal ma anche da goal diversi (eventualmente già eseguiti) e quindi, sostanzialmente, fornire la possibilità di inserire ed utilizzare piani pre-cablati o calcolati al lancio dell'agente.

Effettuando i vari test (Capitolo 3) abbiamo notato che circa il 70% dei piani generati è frutto di copia e non di computazione dell'algoritmo di ricerca.

1.3.3 Abort e ripianificazione: avvicinarsi all'open-minded

Abbiamo già parlato di come funziona il sistema di ripianificazione in seguito all'abort di un goal. Desideriamo però far notare che se ad ogni step abortissimo il goal attuale, l'agente diventerebbe di fatto un agente open-minded. Questo accadrebbe perché, quando si abortisce un goal, l'agente può scegliere di prenderne un altro in considerazione qualora lo ritenga più appetibile.

È evidente che un approccio di questo tipo comporterebbe un significativo aumento del numero di pianificazioni e dunque bisognerebbe valutare un sistema che riduca le risorse utilizzate per la definizione del piano ottimo. Si potrebbe ad esempio impiegare un'euristica per scegliere le service cell più promettenti non garantendo più un piano ottimo per l'obiettivo corrente ma diminuendo la computazione così da permettere all'agente di riconsiderare le sue scelte ad ogni step.

Quanto appena detto è possibile grazie al fatto che l'agente è in grado di pianificare le azioni necessarie per portare a termine un goal a partire da uno stato qualsiasi: sostanzialmente, è in grado di trovare soluzione a *qualsiasi* goal a partire da un *qualsiasi* stato interno.

1.3.4 CheckFinish: un'operazione troppo rischiosa

Il nostro agente non compie mai un'azione di `CheckFinish`. Sono due le ragioni per cui abbiamo deciso di non adoperare mai questa operazione:

1. La `CheckFinish` è da adoperarsi qualora un tavolo non avvisi l'agente di aver terminato. Tuttavia un tavolo che si trovi in questa situazione, non porta ad alcuna penalità e quindi non c'è ragione, dal punto di vista dell'agente, di andare a verificare se il tavolo abbia effettivamente terminato o meno. Anzi, se un tavolo dovesse rimanere in questa situazione fino al termine della computazione sarebbe un vantaggio per l'agente che non avrebbe più ordini da soddisfare da parte del tavolo stesso[2].

[2] Questo avviene in conformità alle regole stabilite nel modulo `ENV`.

Non è quindi premiato il turn-over dei tavoli, principio che sarebbe certamente importante per il proprietario di un locale di ristorazione.

2. L'operazione di `CheckFinish` è estremamente costosa in termini di tempo (e quindi di penalità assunta a causa dell'attesa degli altri tavoli) e comporta anche rischi nel caso in cui il tavolo non abbia effettivamente consumato l'ordine.

1.3.5 Preferire alla `Turn` una `Wait`?

Dal punto di vista della penalità converrebbe attuare *sempre* una azione di `Turnright` o `Turnleft` anziché una di `Wait`, infatti le prime costano un decimo di tempo ma fanno passare comunque uno step che è la nostra unità temporale. Abbiamo però deciso di compiere comunque azioni di `Wait` poiché se l'agente avesse una batteria (come avrebbe in un ambiente reale) le operazioni di `Turn` sarebbero certamente più costose dal punto di vista energetico.

Capitolo 2

Scegliere il goal: le strategie

2.1 Introduzione

La scelta di quale goal esaudire per primo è di fondamentale importanza. Studiando i criteri di utilità abbiamo cercato di sviluppare diverse strategie che rendessero il nostro agente in grado di minimizzare la penalità accumulata. Possiamo dividere le strategie in due grandi insiemi: strategie che fanno uso del criterio di utilità (Full Utility, Penalty on Distance e Pure Penalty) e strategie che invece non ne fanno uso (FIFO, LIFO, Random e Distance).

Le strategie utility based e la strategia Distance fanno inoltre uso dei dati raccolti dal modulo `EURISTIC` parliamo a seguire.

2.1.1 Cosa fa il modulo `EURISTIC`

Sviluppare una strategia che sfrutta l'utilità necessita di dati aggiuntivi relativi alle penalità attribuite durante l'attesa di un goal.

Giacché esiste un numero finito di sequenze di task generabili è stato semplice prevedere la scelta che verrebbe effettuata in pianificazione se il goal preso in esame fosse quello portato in esecuzione. Dato il piano, abbiamo quindi calcolato le distanze che si percorrerebbero e la penalità che si assumerebbe ad ogni passo. Il modulo `EURISTIC` compila quindi un fatto `goal-path` che elenca la serie di places (targets) che coinvolgono il piano, le distanze fra un place e l'altro e la quantità di penalità assunta ad ogni passo nel percorso che collega due places. Il calcolo della penalità è impreciso così come lo è quello delle distanze: nel primo caso si assume l'esecuzione di un'azione di `Forward` ad ogni passo e nell'altro si utilizza la distanza di Manhattan.

Starà alla singola strategia scegliere quali di questi dati usare e come combinarli al fine di individuare il goal che porterà al maggior risparmio di penalità una volta terminato.

Il `goal-path` *non considera* le penalità assunte nel momento in cui si compiono azioni di Load o Delivery: questa semplificazione è dettata da una difficoltà tecnica nel calcolare la quantità precisa di penalità attribuibile ad ogni istante di tempo durante l'esecuzione sequenziale di più azioni di Load/Delivery. Il motivo per cui tale calcolo non è stato considerato risiede nella complessità tecnica della valutazione dell'esatta quantità di penalità assegnata durante l'esecuzione di ogni azione: infatti è l'esecuzione di un'azione stessa (nel caso di `Deliver`) a comportare un cambiamento nel valore totale di penalità acquisito.

2.2 FIFO e LIFO

Le prime due strategie che abbiamo implementato sono state FIFO e LIFO.

La strategia LIFO sfrutta il funzionamento di CLIPS per cui le regole sono attivate dai fatti più recenti. Giacché i goal vengono creati non appena arriva un ordine o l'agente si sporca, il fatto `goal` più recente sarà sempre quello relativo all'evento accaduto più tardi nel tempo. Fanno eccezione i goal abortiti che, a causa dell'aggiornamento dello slot `status`, possono risultare i più recenti e dunque essere scelti per primi allo step successivo dopo la loro dichiarazione di abortiti. La strategia FIFO considera i goal secondo l'ordine di arrivo.

Entrambe le strategie non sembrano essere vantaggiose per questo tipo di agente: la penalità attribuita all'agente non dipende infatti soltanto da quanto tempo un'ordine debba attendere ma piuttosto dipende da quanto l'ordine è corposo e queste strategie non tengono conto di questo dato.

2.3 Random

La strategia Random sceglie i goal senza seguire alcun criterio: la scelta è guidata dalla generazione di un numero casuale.

Questa strategia può portare ad un'esecuzione dei goal inaspettata, che porta l'agente a preferire goal apparentemente poco interessanti: di fatto, la strategia Random in alcune esecuzioni sembra poter ''guardare al futuro'' schivando volontariamente la scelta ottimale. Nella maggior parte dei casi queste scelte portano ad ottenere alti valori di penalità, ma in determinate ed imprevedibili situazioni la strategia Random può dare luogo a scelte capaci di ridurre in modo drastico le penalità. È evidente che questo tipo di evoluzioni sia guidato dal caso: studiare quindi una singola esecuzione con strategia Random risulta poco significativo ma studiarne una moltitudine può dire molto in merito all'ambiente.

La strategia Random è molto intuitiva e non richiede alcuno sforzo da parte

dell'agente; per questo motivo può risultare utile come benchmark tramite il quale paragonare l'effettivo vantaggio dell'impiego di una strategia piuttosto che un'altra.

2.4 Distance

Distance è la prima strategia che abbiamo implementato che fa uso dei dati forniti da `EURISTIC`. Gli unici dati impiegati sono in merito alla distanza (si ignorano quindi le penalità).

L'idea base è molto semplice: si porta a termine per primo il goal la cui somma delle distanze da percorrere è minore. La formula usata è quindi:

$$d_{0,1} + d_{1,2} + d_{2,3} + d_{3,4} + d_{4,5}$$

dove con d indichiamo un valore di distanza e a pedice indichiamo l'identificativo dei place coinvolti. Ad esempio $d_{0,1}$ è la distanza che intercorre tra il place 0 ed il place 1.
Nel caso in cui il goal necessiti di un numero di place minore di cinque le distanze relative ai place non coinvolti saranno pari a zero.

2.5 Pure Penalty

La strategia Pure Penalty sfrutta solamente i dati relativi alle penalità forniti dal modulo `EURISTIC`. Ricordiamo che i valori restituiti rappresentano la quantità di penalità per passo e dunque se deconstualizzati dalla distanza possono risultare poco significativi. Abbiamo comunque deciso di implementare questa strategia poiché il calcolo della penalità al passo è decisamente più immediato rispetto al calcolo delle distanze. Si noti inoltre che l'uso delle penalità in maniera ''pura'' permette di non essere soggetti all'errore di calcolo dovuto all'utilizzo della distanza di Manhattan. La formula usata è quindi:

$$p_{0,1} + p_{1,2} + p_{2,3} + p_{3,4} + p_{4,5}$$

Il significato della formula è simile a quanto descritto per la strategia Distance, ma in questo caso con p indichiamo un valore di penalità.

2.6 Penalty on Distance

La strategia Penalty on Distance sfrutta tutte le informazioni calcolate dal modulo `EURISTIC`. Se con la strategia Distance avevamo considerato soltanto le distanze e con la PP soltanto le penalità, PoD cerca di combinare i due dati

attribuendo un corretto peso alle penalità e cercando quindi una soluzione più fine. La formula usata è:

$$\frac{p_{0,1} + p_{1,2} + p_{2,3} + p_{3,4} + p_{4,5}}{d_{0,1} + d_{1,2} + d_{2,3} + d_{3,4} + d_{4,5}}$$

2.7 Full Utility

La strategia Full Utility si discosta di poco dalla PoD ma cerca di pesare ogni distanza percorsa per la penalità che durante quella distanza l'agente assumerà. Aumentare l'uso delle distanze nella formula vuol dire renderla più sensibile agli errori derivanti dall'impiego della distanza di Manhattan. La formula usata è quindi:

$$\frac{(p_{0,1} \cdot d_{01}) + (p_{1,2} \cdot d_{1,2}) + (p_{2,3} \cdot d_{2,3}) + (p_{3,4} \cdot d_{3,4}) + (p_{4,5} \cdot d_{3,4})}{d_{0,1} + d_{1,2} + d_{2,3} + d_{3,4} + d_{4,5}}$$

Capitolo 3

Fase di sperimentazione

3.1 Metodo di confronto

Per testare[1] l'agente abbiamo sviluppato sette diversi environment (di cui parleremo nel dettaglio nella Sezione 3.2) cercando di rappresentare un insieme di situazioni il più eterogeneo possibile.

Avendo sviluppato sette diverse strategie abbiamo preferito non testare ogni strategia su ogni environment ma abbiamo scelto i tre più rappresentativi così da poter ottenere una sorta di ''classifica'' delle strategie, per poi testare i rimanenti quattro environment sulle migliori due della classifica (Sezione 3.5).

Una volta definita la migliore strategia abbiamo implementato alcune ulteriori migliorie (trattate nella Sezione 3.6) al fine di misurarne l'impatto sulla penalità e valutarne le conseguenze sui tempi di esecuzione.

3.2 Gli environment

Le tre variabili sulle quali ci siamo mossi per delineare gli environment sono la mappa, la configurazione degli ordini (in termini di quantità e frequenza) e il numero di persone in movimento.

Le mappe

Le mappe *map1* e *map2*, di dimensione 10×10, sono quelle fornite dal docente (Figura 3.1) La terza mappa (17×17) è stata creata ad hoc per analizzare il comportamento dell'agente in ambienti più ampi (Figura 3.2).

[1] I test sono stati eseguiti su una macchina con CPU Intel Core i5 @ 2.70GHz, RAM 8GB con SO Mac OS X 64bit.

Figura 3.1: Le mappe *map1* e *map2*.

Figura 3.2: La mappa *map3*.

Gli ordini

Per quanto concerne gli ordini abbiamo distinto tre livelli di difficoltà, basati sia sulla loro frequenza che sulla quantità di cibo/bevande richiesti.

La penalità minima

Combinando mappe e storie abbiamo definito i seguenti ambienti:

Ambiente	Mappa	Difficoltà ordini	Persone
env1	*map1*	semplice	1
env2	*map1*	medio	2
env3	*map1*	difficile	3
env4	*map2*	medio	2
env5	*map2*	difficile	3
env6	*map3*	medio	6
env7	*map3*	difficile	6

Per poter valutare i punteggi di penalità ottenuti dalle varie strategie abbiamo cercato di calcolare la penalità minima (ovvero l'ottimo), la penalità massima e la penalità media. Per fare ciò abbiamo compiuto un numero di esecuzioni (con strategia random) tali da avere un quantità sufficiente[2] di casistiche su cui ponderare statistiche legittime.

Ecco i risultati (con N.V.D. intendiamo il numero di valori distinti di penalità che abbiamo ottenuto):

Ambiente	N. Esecuzioni	N. V. Distinti	P. avg	P. min	P. max
env1	1.000	1	431	431	431
env2	1.000	39	4.489	3.991	5.220
env3	15.000	3.132	13.564	10.894	16.868
env4	500	38	3.581	2.924	4.451
env5	5.000	509	9.585	6.134	85.063
env6	7.000	4.838	33.630	27.357	49.241
env7	8.000	4.769	37.830	19.847	110.692

Gli ambienti che abbiamo deciso di utilizzare per la prima fase di sperimentazione sono l'*env2*, l'*env4* e l'*env6*. Questi infatti coprono gran parte delle problematiche più classiche che l'agente può incontrare senza penalizzare o favorire particolari strategie.

3.3 Test di correttezza: env1

La sperimentazione sull'*env1* è particolare: c'è un solo ordine ed esiste dunque un solo modo di portare a termine i vari goal che vengono generati. Inoltre la sua esecuzione porta ad un `bump`). L'*env1* funge quindi da sorta di test di correttezza e pertanto ne riportiamo soltanto i dati statistici (che sono uguali per ogni strategia):

[2] Per quanto concerne l'*env6* e l'*env7* abbiamo molti valori distinti rispetto alle esecuzioni poiché ognuna di esse impiega circa 30 secondi e dunque è stato complesso averne un numero tale da poter ritenere la statistica fedele. Possiamo però immaginare che il valore medio di penalità non si discosti più di tanto da quello trovato nelle migliaia di esecuzioni effettuate.

Piani totali	47 (di cui 16,9% copiati)
Ripianificazioni	1
Tempo di esecuzione	1,382 secondi
Penalità totale	341
Fatti generati	30.772
Tempo di attesa medio	59 unità

3.4 Fase 1: test sui primi tre env

3.4.1 Sperimentazione sull'env2

FIFO

Classica politica di scheduling: i goal vengono serviti in ordine di arrivo.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.3.

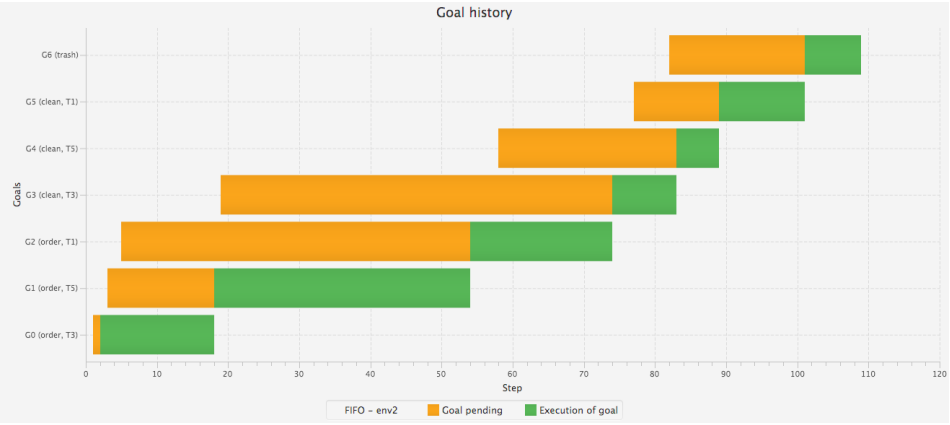

Figura 3.3: Scheduling dei goal con strategia FIFO.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	230 (di cui 76,2% copiati)
Ripianificazioni	0
Tempo di esecuzione	2,005 secondi
Penalità totale	3.991
Fatti generati	75.253
Tempo di attesa medio	125 unità

Andamento della penalità. Ogni esecuzione è caratterizzata dall'andamento delle penalità. Abbiamo due grafici che rappresentano rispettivamente la funzione cumulativa di penalità e l'incremento di penalità per ogni step (Figura 3.4). Non riporteremo tali grafici per ogni esecuzione a meno di notare

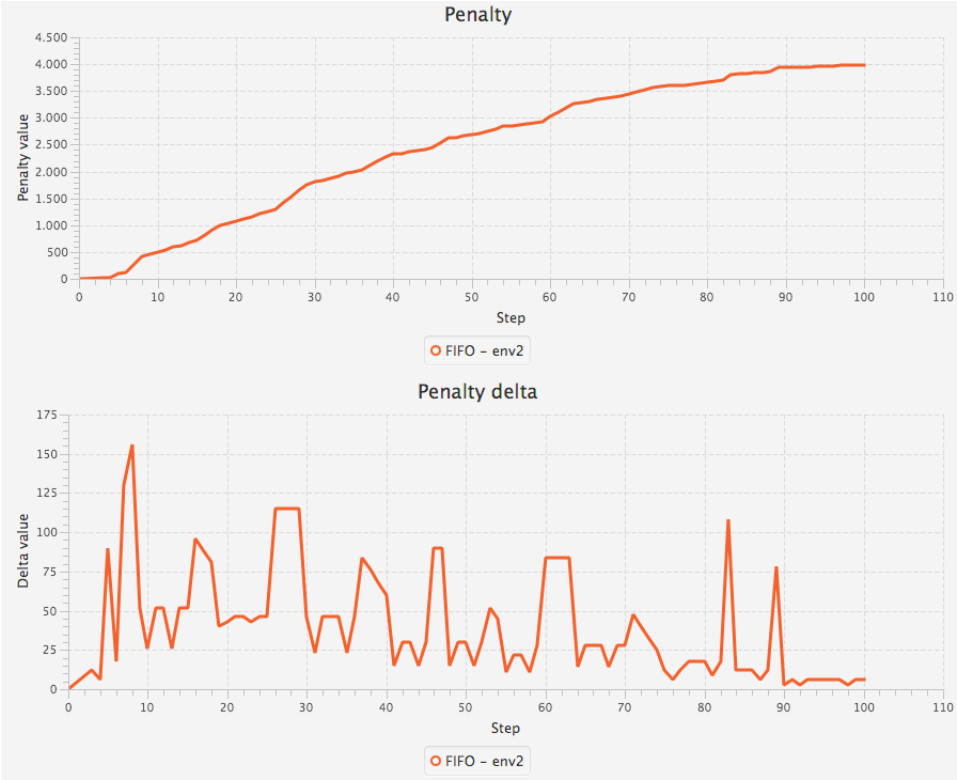

Figura 3.4: Andamento delle penalità (cumulativa e delta).

fenomeni particolarmente interessanti, ma mostreremo soltanto un confronto finale al termine di ogni paragrafo per ogni env testato.
In generale possiamo notare la presenza di picchi nel grafico degli incrementi: giacché sull'asse delle x abbiamo gli `step` e non il tempo, la presenza di un picco indica l'esecuzione di un'azione che occupa molto tempo e quindi in uno step soltanto si accumula molta penalità (si vedano ad esempio gli ultimi due picchi causati da operazioni di `clean`).

Quando invece osserviamo un valore costante di delta possiamo dedurre che si siano compiute azioni di uguale costo temporale e che nessun altro evento abbia modificato la quantità di penalità per step dovuta ai goal pendenti. Un esempio di tale situazione si riscontra agli step 59-62 in cui avvengono quattro `LoadDrink`.

LIFO

Classica politica di scheduling: i goal vengono serviti in ordine inverso rispetto al loro arrivo.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.5.

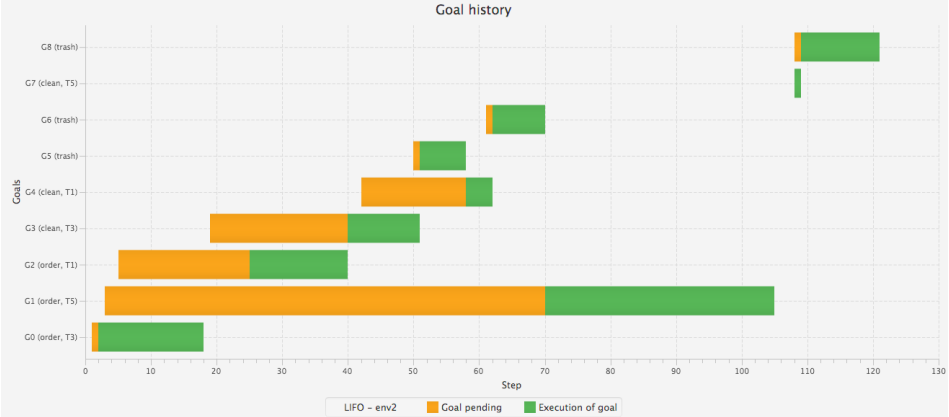

Figura 3.5: Scheduling dei goal con strategia LIFO.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	253 (di cui 68,4% copiati)
Ripianificazioni	1
Tempo di esecuzione	2,080 secondi
Penalità totale	5.164
Fatti generati	78.058
Tempo di attesa medio	147 unità

Random

La strategia random sceglie un goal a caso fra quelli effettuabili.

Per quanto concerne la strategia random, abbiamo effettuato 700 esecuzioni e abbiamo così ottenuto un valore medio di penalità pari a 4.489. A titolo espositivo riportiamo uno di questi casi.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.6.

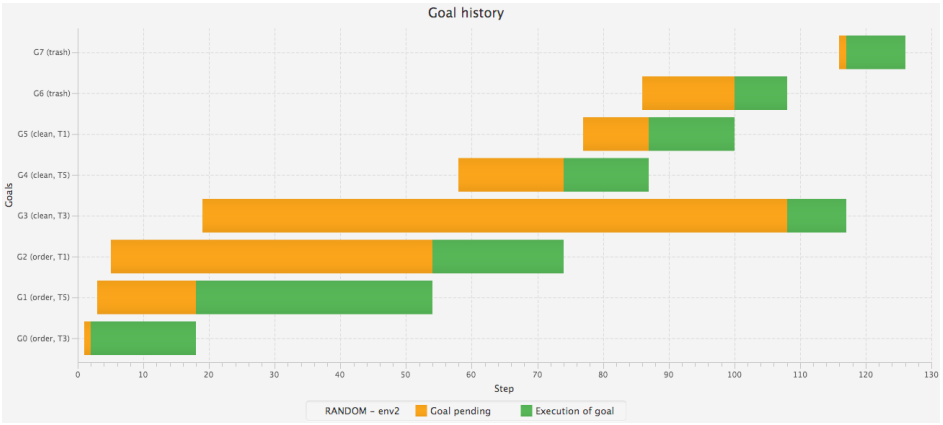

Figura 3.6: Scheduling dei goal con strategia Random.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	248 (di cui 69% copiati)
Ripianificazioni	0
Tempo di esecuzione	2,039 secondi
Penalità totale	4.249
Fatti generati	82.414
Tempo di attesa medio	125 unità

Distance

La strategia distance sceglie il goal che ha distanza di Manhattan minore tra tutti i goal pendenti.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.7.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	253 (di cui 68,4% copiati)
Ripianificazioni	0
Tempo di esecuzione	2,017 secondi
Penalità totale	5.164
Fatti generati	78.212
Tempo di attesa medio	147 unità

Pure Penalty

La strategia PP fa parte delle strategie basate sull'utilità. Viene scelto il goal il cui conseguimento porta alla maggiore diminuzione di penalità.

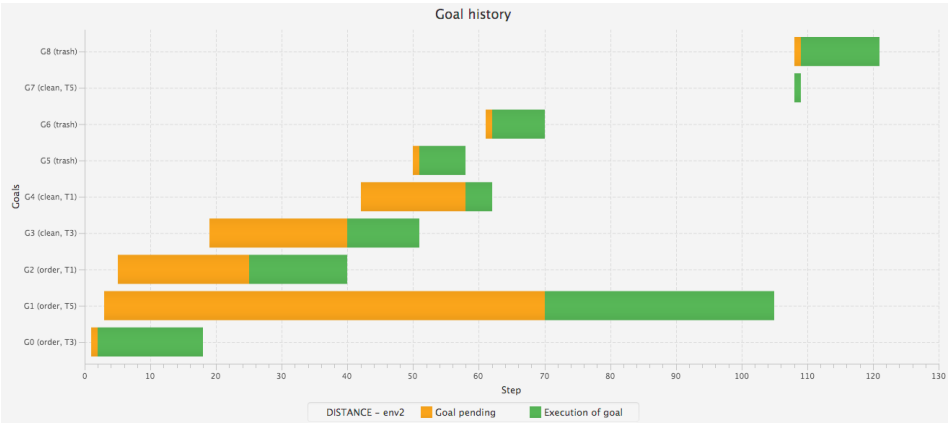

Figura 3.7: Scheduling dei goal con strategia Distance.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.8.

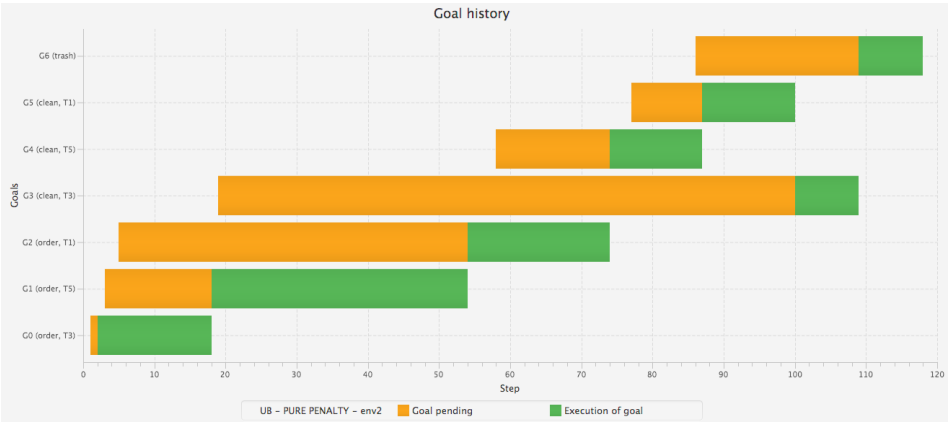

Figura 3.8: Scheduling dei goal con strategia PP.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	230 (di cui 76,2% copiati)
Ripianificazioni	0
Tempo di esecuzione	2,032 secondi
Penalità totale	4.183
Fatti generati	77.058
Tempo di attesa medio	125 unità

Penalty on Distance

La strategia PoD fa parte delle strategie basate sull'utilità. Si comporta come la PP ma il valore di penalità è diviso per la distanza (calcolata come nella strategia Distance).

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.9.

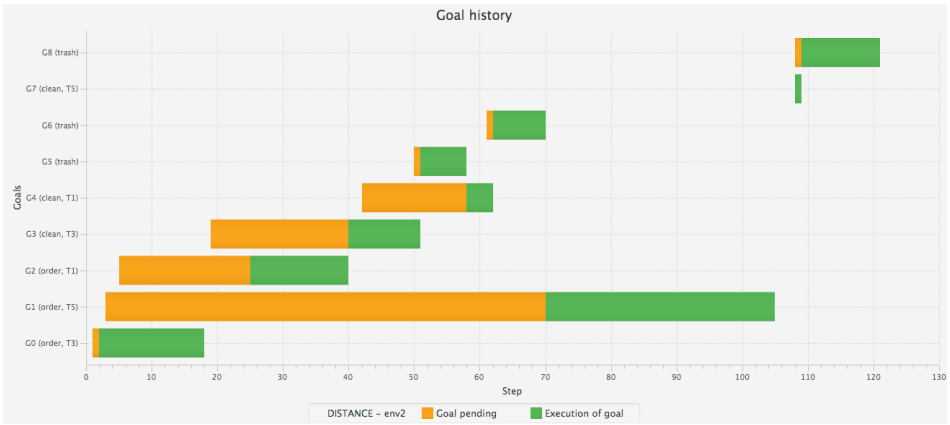

Figura 3.9: Scheduling dei goal con strategia PoD.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	233 (di cui 71,2% copiati)
Ripianificazioni	2
Tempo di esecuzione	1,963 secondi
Penalità totale	4.529
Fatti generati	74.763
Tempo di attesa medio	112 unità

Full Utility

La strategia FU fa parte delle strategie basate sull'utilità. I valori di distanza appaiono anche al numeratore per pesare la penalità assegnate ad ogni goal.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.10.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

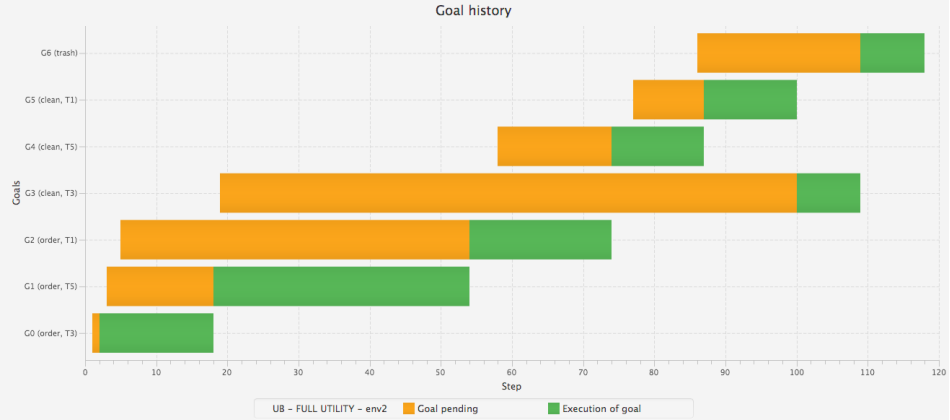

Figura 3.10: Scheduling dei goal con strategia FU.

Piani totali	230 (di cui 76,2% copiati)
Ripianificazioni	0
Tempo di esecuzione	2,037 secondi
Penalità totale	4.183
Fatti generati	77.058
Tempo di attesa medio	125 unità

3.4.2 Comparazione dei risultati sull'env2

Dati statistici. I dati statistici ottenuti in merito all'*env2* sono i seguenti:

Strategia	Penalità	Distanza dalla media
FIFO	**3.991**	-498
Full Utility	4.183	-306
Pure Penalty	4.183	-306
Random	4.489	-
Penalty on Distance	4.529	+40
Distance	5.164	+675
LIFO	5.164	+675

La strategia vincente è stata FIFO, unica strategia ad aver ottenuto la penalità minima. Dal punto di vista dell'attesa media dei tavoli è invece PoD ad avere la meglio. Si evince quindi che far aspettare di meno ogni tavolo non necessariamente porta ad ottenere una penalità minore: si ritiene quindi migliore una strategia che riesca a lasciare pendenti quegli ordini che sono meno costosi dal punto di vista della penalità.
Notiamo infatti che le strategie Full Utility e Pure Penalty, che mirano ad implementare il principio appena esposto, totalizzano un basso punteggio di penalità rispetto alle ben più costose LIFO, Random e Distance.

La vittoria della stragia FIFO stupisce: almeno dal punto di vista intuitivo, FIFO non può essere una strategia vincente in ogni caso (starà alle fasi future della sperimentazione dimostrarlo) e quindi giustifichiamo il suo risultato ritenendo che la sequenza di ordini con la quale è configurato l'*env2* porti ad un corretto scheduling da parte di FIFO.

Andamento della penalità. Il grafico comparativo delle penalità per quanto concerne l'*env2* è mostrato in Figura 3.11.

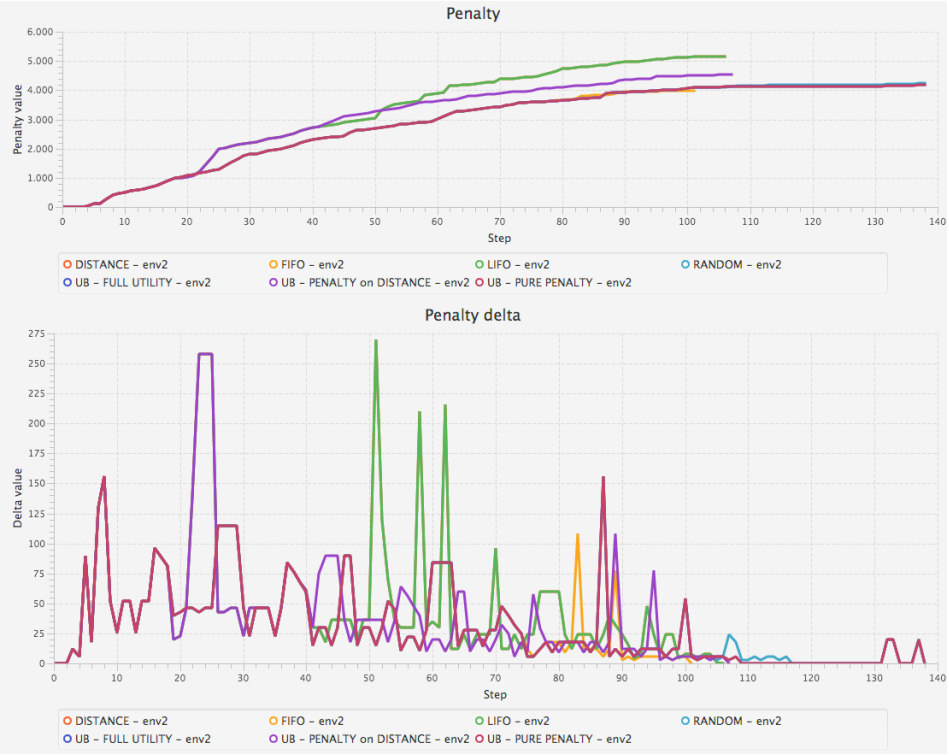

Figura 3.11: Grafico comparativo delle penalità.

Come si nota abbiamo due coppie di serie cumulative sovrapposte: la FU coincide con la PP mentre la Distance coincide con la LIFO. Notiamo inoltre che il caso preso in considerazione per la strategia Random è molto vicino all'andamento della PP/FU.
Più interessante è il grafico che compara le delta-penalità. Notiamo come i primi step diano luogo a delta coincidenti per ogni strategia: questo avviene perché c'è soltanto un ordine in attesa e quindi ogni strategia non può che scegliere quello. Con il proseguire degli step vediamo che ogni strategia inizia a compiere scelte diverse fra i goal pendenti e quindi a seguire un andamento caratteristico. Possiamo notare la presenza di picchi e di zone di delta costante come già evidenzato in precedenza a inizio paragrafo.

Osserviamo inoltre a fondo grafico la presenza di due picchi (del valore di 20 penalità) che indicano il fatto che il nostro agente, fermo poiché non ci sono più goal, abbia ostruito il cammino di una persona e quindi si sia spostato per lasciarla passare utilizzando la closeness detection(Sezione 1.3.1).

Tempo d'attesa medio dei tavoli. Riportiamo anche una comparazione fra i tempi medi d'attesa dei tavoli in Figura 3.12.

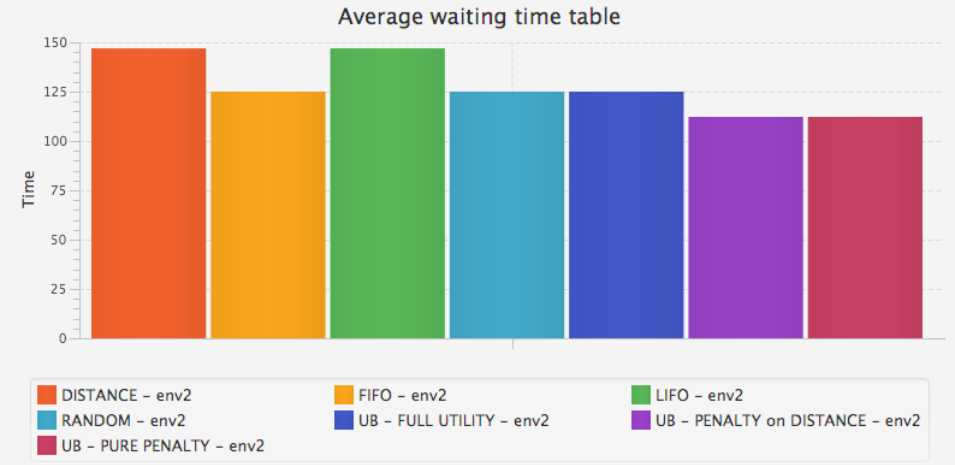

Figura 3.12: Grafico comparativo dei tempi d'attesa per l'*env2*.

3.4.3 Sperimentazione sull'env4

FIFO

Classica politica di scheduling: i goal vengono serviti in ordine di arrivo.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.13.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	53 (di cui 49,1% copiati)
Ripianificazioni	1
Tempo di esecuzione	1,916 secondi
Penalità totale	4.070
Fatti generati	55.548
Tempo di attesa medio	111,33 unità

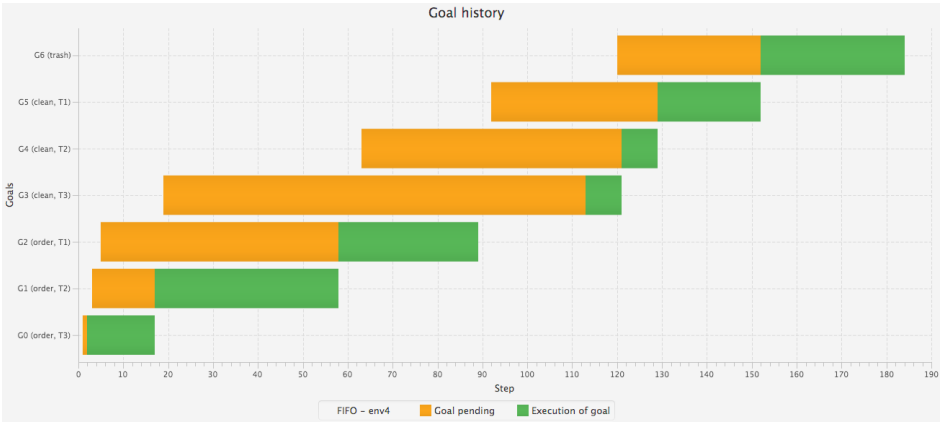

Figura 3.13: Scheduling dei goal con strategia FIFO.

LIFO

Classica politica di scheduling: i goal vengono serviti in ordine inverso rispetto al loro arrivo.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.14.

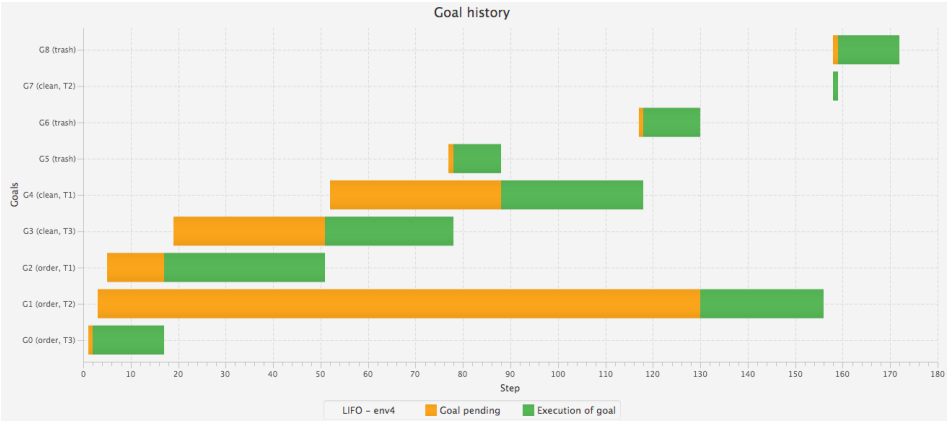

Figura 3.14: Scheduling dei goal con strategia LIFO.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	56 (di cui 46,4% copiati)
Ripianificazioni	0
Tempo di esecuzione	1,777 secondi
Penalità totale	3.588
Fatti generati	55.390
Tempo di attesa medio	160,33 unità

Random

La strategia random sceglie un goal a caso fra quelli effettuabili.

Per quanto concerne la strategia random, abbiamo effettuato 500 esecuzioni e abbiamo così ottenuto un valore medio di penalità pari a 3.581. A titolo espositivo riportiamo uno di questi casi.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.15.

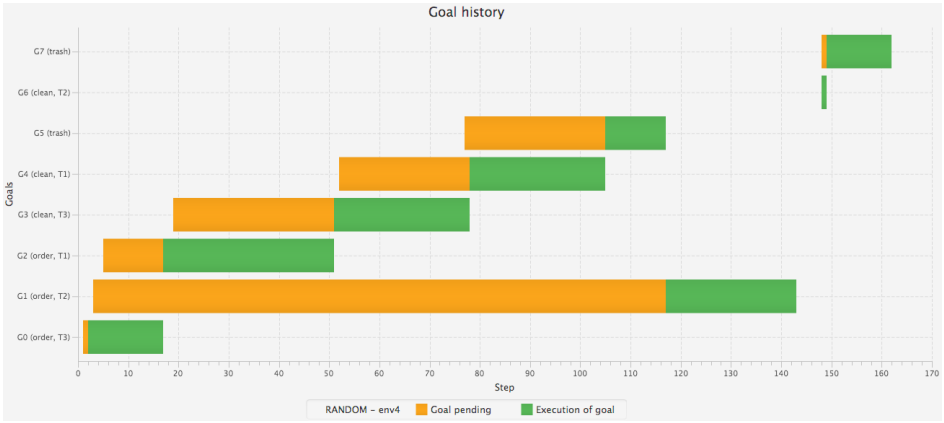

Figura 3.15: Scheduling dei goal con strategia Random.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	54 (di cui 48,1% copiati)
Ripianificazioni	0
Tempo di esecuzione	2,121 secondi
Penalità totale	3.354
Fatti generati	53.001
Tempo di attesa medio	151,67 unità

Distance

La strategia distance sceglie il goal che ha distanza di Manhattan minore tra tutti i goal pendenti.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.16.

Figura 3.16: Scheduling dei goal con strategia Distance.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	56 (di cui 46,4% copiati)
Ripianificazioni	0
Tempo di esecuzione	2,071 secondi
Penalità totale	3.588
Fatti generati	55.548
Tempo di attesa medio	160,33 unità

Pure Penalty

La strategia PP fa parte delle strategie basate sull'utilità. Viene scelto il goal il cui conseguimento porta alla maggiore diminuzione di penalità.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.17.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

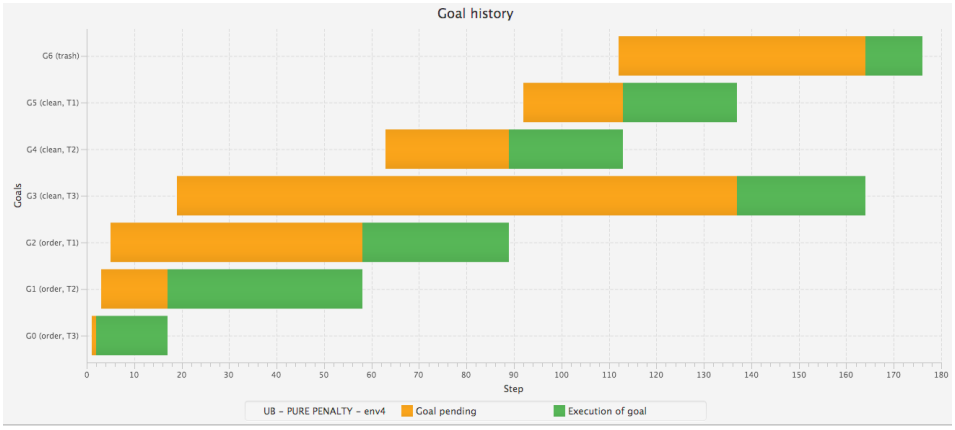

Figura 3.17: Scheduling dei goal con strategia PP.

Piani totali	52 (di cui 50% copiati)
Ripianificazioni	0
Tempo di esecuzione	1,815 secondi
Penalità totale	4.163
Fatti generati	55.724
Tempo di attesa medio	111,33 unità

Penalty on Distance

La strategia PoD fa parte delle strategie basate sull'utilità. Si comporta come la PP ma il valore di penalità è diviso per la distanza (calcolata come nella strategia Distance).

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.18.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	52 (di cui 50% copiati)
Ripianificazioni	0
Tempo di esecuzione	1,770 secondi
Penalità totale	2.924
Fatti generati	50.230
Tempo di attesa medio	105,66 unità

Full Utility

La strategia FU fa parte delle strategie basate sull'utilità. I valori di distanza appaiono anche al numeratore per pesare la penalità assegnate ad ogni goal.

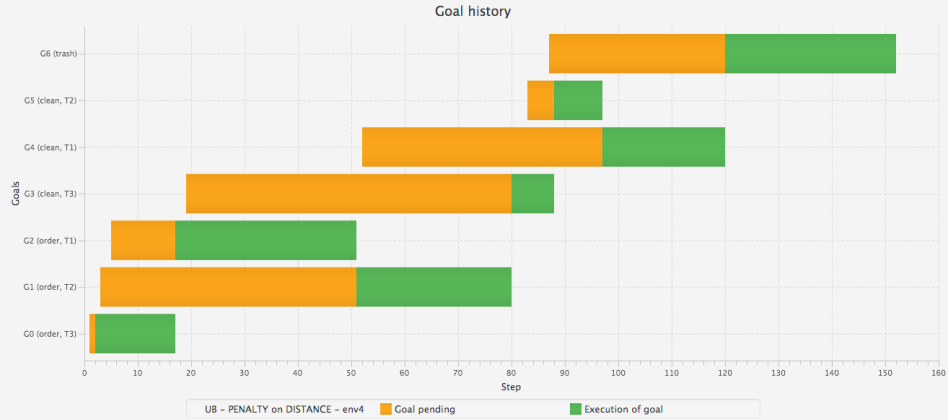

Figura 3.18: Scheduling dei goal con strategia PoD.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.19.

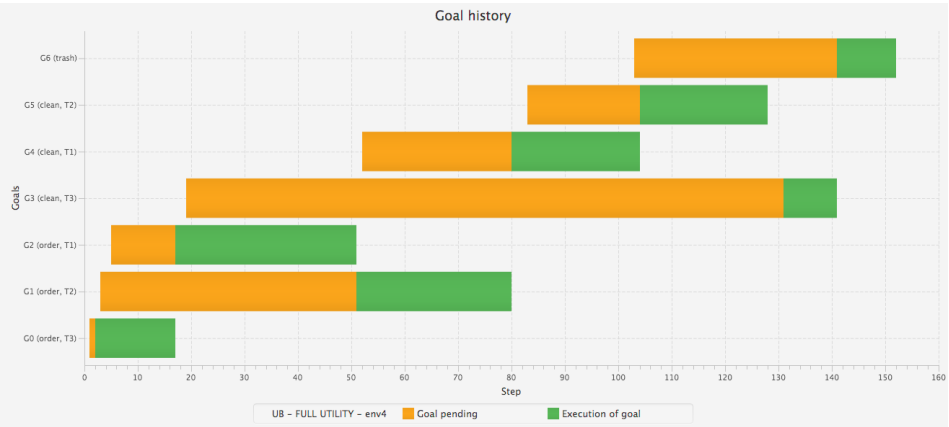

Figura 3.19: Scheduling dei goal con strategia FU.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	53 (di cui 49,1% copiati)
Ripianificazioni	1
Tempo di esecuzione	1,824 secondi
Penalità totale	3.370
Fatti generati	51.805
Tempo di attesa medio	105,66 unità

3.4.4 Comparazione dei risultati sull'env4

Dati statistici. I dati statistici ottenuti in merito all'*env4* sono i seguenti:

Strategia	Penalità	Distanza dalla media
Penalty on Distance	**2.924**	-657
Full Utility	3.370	-211
Random	3.581	-
LIFO	3.588	+7
Distance	3.588	+7
FIFO	4.070	+489
Pure Penalty	4.163	+582

La strategia che si è rivelata vincente è la PoD che è riuscita a raggiungere l'ottimo. In seconda posizione, così come già avvenuto nell'*env2*, si trova la FU. Le strategie LIFO e Distance si piazzano intorno alla media statistica mentre invece si comportano male le strategie di PP e FIFO. Stupisce il comportamento di PP, che solitamente si comporta bene: cercheremo di capire dal grafico delle penalità la ragione di questo punteggio così deludente.

Andamento della penalità. Il grafico comparativo delle penalità per quanto concerne l'*env4* è il seguente:

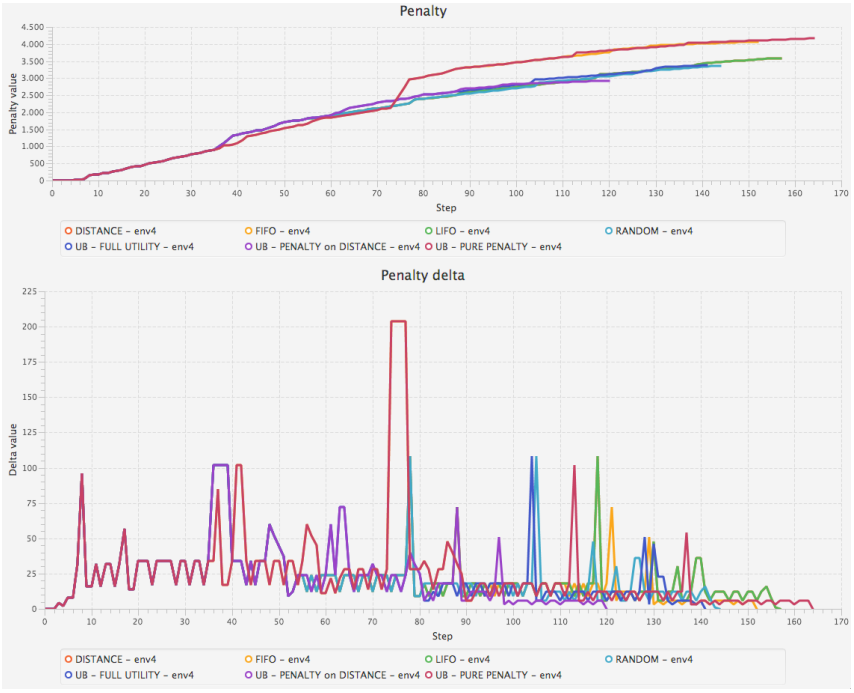

Figura 3.20: Grafico comparativo delle penalità.

L'unica sovrapposizione totale riguarda le strategie LIFO e Distance.
Osservando la curva delta-penalità di PP notiamo un grosso picco, dovuto, come già notato in altri casi, a quattro `LoadDrink`. Un calcolo più preciso da parte del modulo `EURISTIC` della penalità assunta in fase di Load avrebbe potuto guidare la strategia in maniera più previdente modificando (e sperabilmente diminuendo) la quantità di penalità assunta in fase di Load.

Anche in questo caso le curve della delta penalità si vanno a dividere con il proseguire dell'esecuzione.

Tempo d'attesa medio. Riportiamo anche una comparazione fra i tempi medi d'attesa in Figura 3.21.

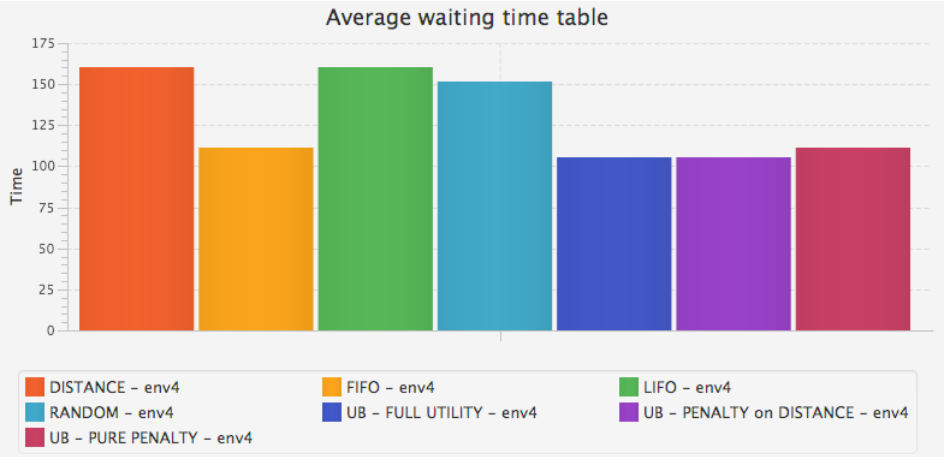

Figura 3.21: Grafico comparativo dei tempi d'attesa per l'*env4*.

3.4.5 Sperimentazione sull'env6

FIFO

Classica politica di scheduling: i goal vengono serviti in ordine di arrivo.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.22.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

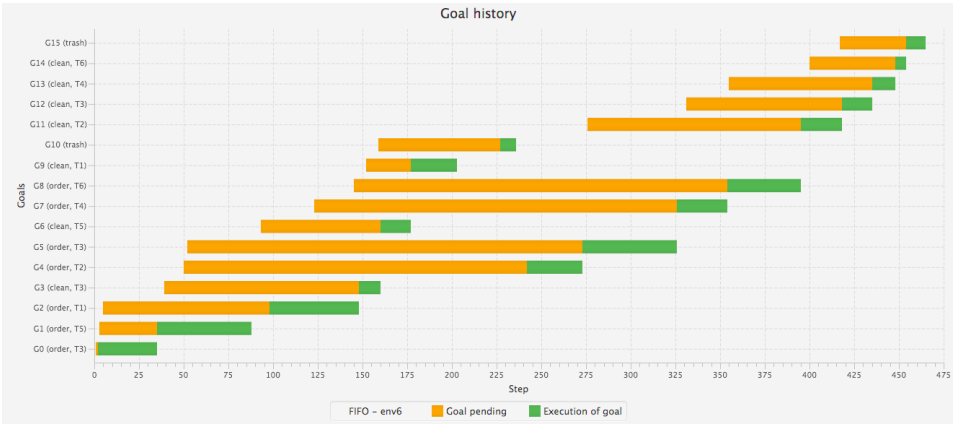

Figura 3.22: Scheduling dei goal con strategia FIFO.

Piani totali	1.627 (di cui 78,2% copiati)
Ripianificazioni	3
Tempo di esecuzione	37,636 secondi
Penalità totale	34.482
Fatti generati	1.531.943
Tempo di attesa medio	436,43 unità

LIFO

Classica politica di scheduling: i goal vengono serviti in ordine inverso rispetto al loro arrivo.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.23.

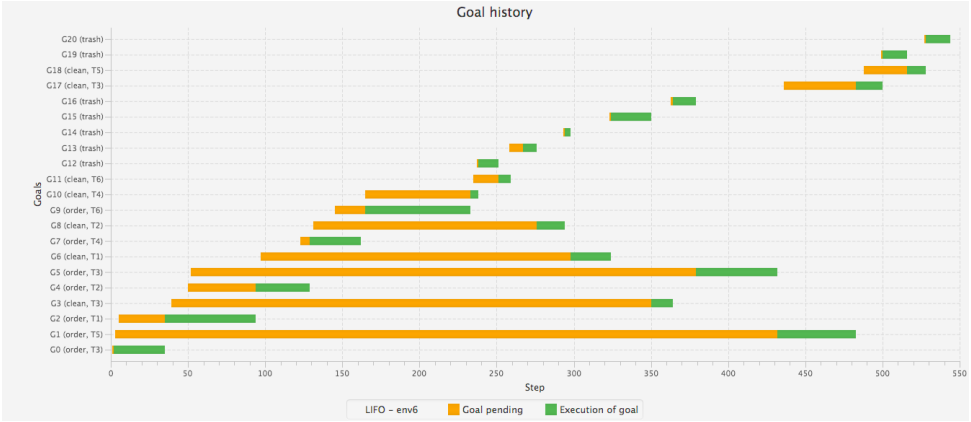

Figura 3.23: Scheduling dei goal con strategia LIFO.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	2.151 (di cui 74,8% copiati)
Ripianificazioni	2
Tempo di esecuzione	56,580 secondi
Penalità totale	33.258
Fatti generati	2.453.692
Tempo di attesa medio	423,14 unità

Random

La strategia random sceglie un goal a caso fra quelli effettuabili.

Per quanto concerne la strategia random, abbiamo effettuato 7.000 esecuzioni e abbiamo così ottenuto un valore medio di penalità pari a 33.630. A titolo espositivo riportiamo uno di questi casi.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.24.

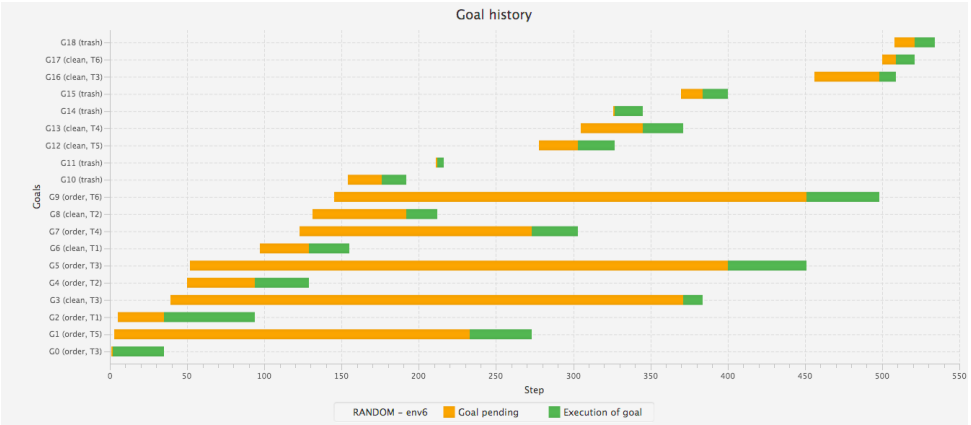

Figura 3.24: Scheduling dei goal con strategia Random.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	2.296 (di cui 76,6% copiati)
Ripianificazioni	3
Tempo di esecuzione	52,411 secondi
Penalità totale	39.252
Fatti generati	2.208.515
Tempo di attesa medio	486,43 unità

Distance

La strategia distance sceglie il goal che ha distanza di Manhattan dall'agente minore tra tutti i goal pendenti.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.25.

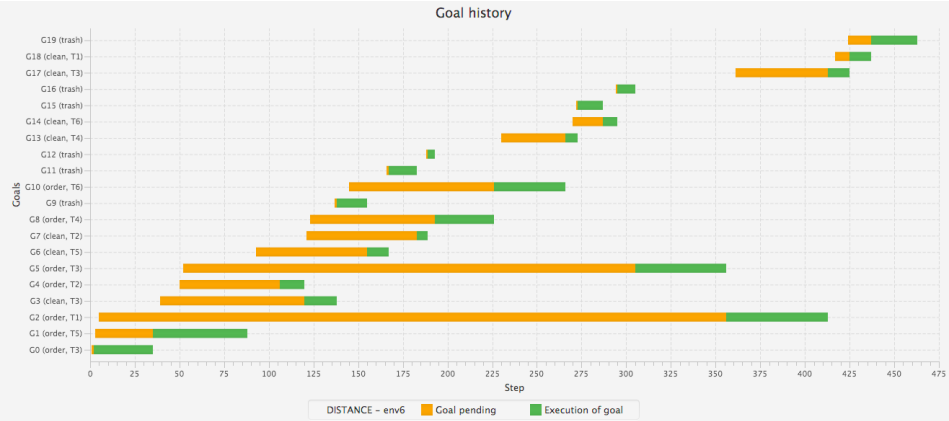

Figura 3.25: Scheduling dei goal con strategia Distance.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	1.686 (di cui 75,3% copiati)
Ripianificazioni	1
Tempo di esecuzione	40,786 secondi
Penalità totale	30.141
Fatti generati	1.651.160
Tempo di attesa medio	425,57 unità

Pure Penalty

La strategia PP fa parte delle strategie basate sull'utilità. Viene scelto il goal il cui conseguimento porta alla maggiore diminuzione di penalità.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.26.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Figura 3.26: Scheduling dei goal con strategia PP.

Piani totali	1.532 (di cui 79,2% copiati)
Ripianificazioni	0
Tempo di esecuzione	36,299 secondi
Penalità totale	29.548
Fatti generati	1.373.041
Tempo di attesa medio	377,57 unità

Penalty on Distance

La strategia PoD fa parte delle strategie basate sull'utilità. Si comporta come la PP ma il valore di penalità è diviso per la distanza (calcolata come nella strategia Distance).

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.27.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	1.617 (di cui 77,1% copiati)
Ripianificazioni	3
Tempo di esecuzione	39,641 secondi
Penalità totale	28.673
Fatti generati	1.569.802
Tempo di attesa medio	353,29 unità

Full Utility

La strategia FU fa parte delle strategie basate sull'utilità. I valori di distanza appaiono anche al numeratore per pesare la penalità assegnate ad ogni goal.

Figura 3.27: Scheduling dei goal con strategia PoD.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.28.

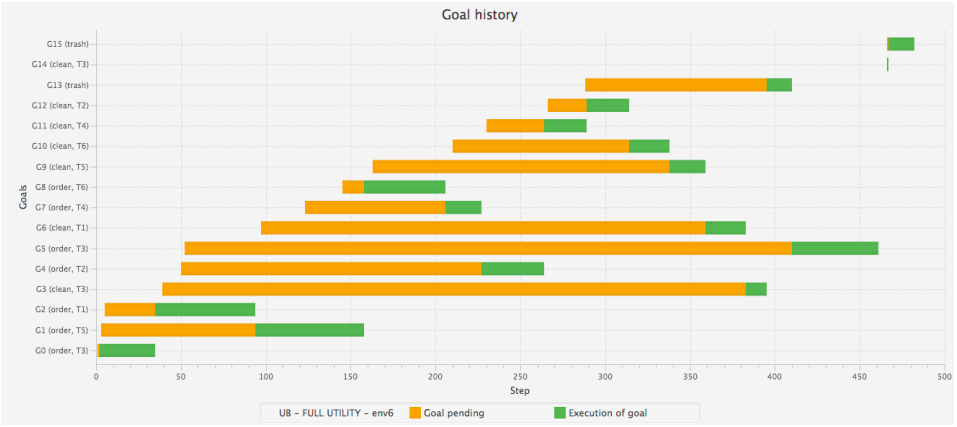

Figura 3.28: Scheduling dei goal con strategia FU.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	1.532 (di cui 79,2% copiati)
Ripianificazioni	0
Tempo di esecuzione	34,807 secondi
Penalità totale	29.548
Fatti generati	1.373.041
Tempo di attesa medio	377,57 unità

3.4.6 Comparazione dei risultati sull'env6

Dati statistici. I dati statistici ottenuti in merito all'*env6* sono i seguenti:

Strategia	Penalità	Distanza dalla media
Penalty on Distance	**28.673**	-4.957
Full Utility	29.548	-4.082
Pure Penalty	29.548	-4.082
Distance	30.141	-3.489
LIFO	33.258	-372
Random	33.630	-
FIFO	34.482	+852

Ancora una volta è PoD la strategia vincente, mentre le altre posizioni alte sono conquistate dalle strategie che impiegano l'utilità come criterio di scelta.

Andamento della penalità. Il grafico comparativo delle penalità per quanto concerne l'*env6* è il seguente:

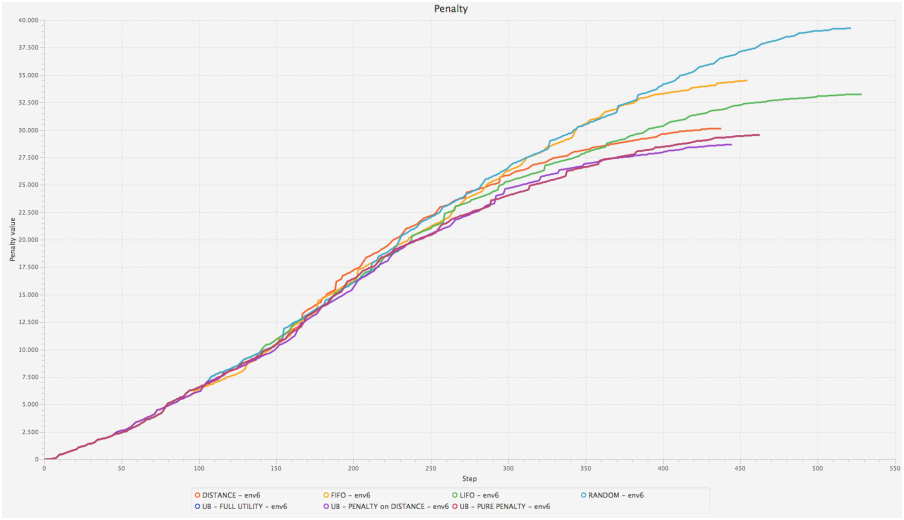

Figura 3.29: Grafico comparativo delle penalità.

Abbiamo una sovrapposizione totale fra le strategie FU e PP.
Rispetto agli altri environment, notiamo che c'è una sorta di raggruppamento delle curve: strategie che non fanno uso del criterio di utilità hanno curve più alte rispetto alle strategie che ne fanno uso. Questo fattore può essere causato dall'aumento delle distanze reciproche fra gli oggetti della mappa. Anche il grafico delle delta penalità (Figura 3.30) mostra picchi molto pronunciati solamente nelle strategie che non fanno uso di utilità.

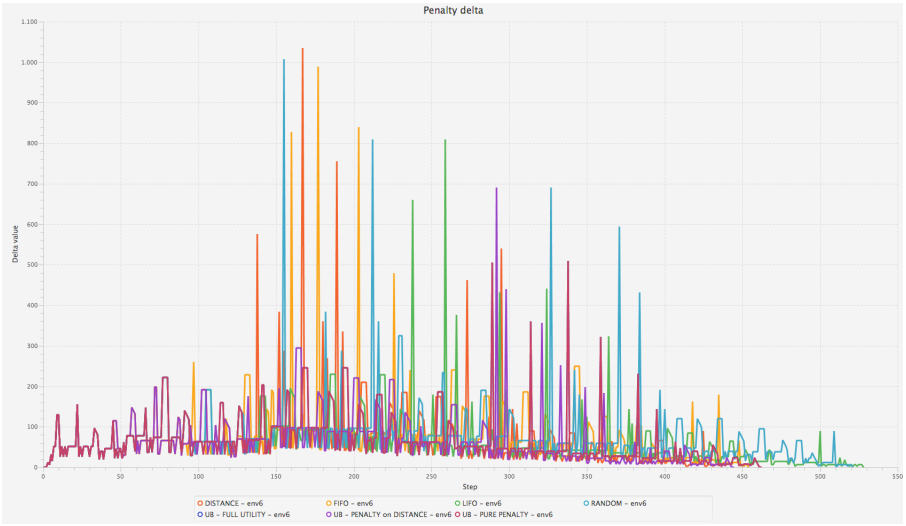

Figura 3.30: Grafico comparativo delle delta-penalità.

Tempo d'attesa medio. Riportiamo anche una comparazione fra i tempi medi d'attesa in Figura 3.31. Anche in questo caso notiamo due 'insiemi' di

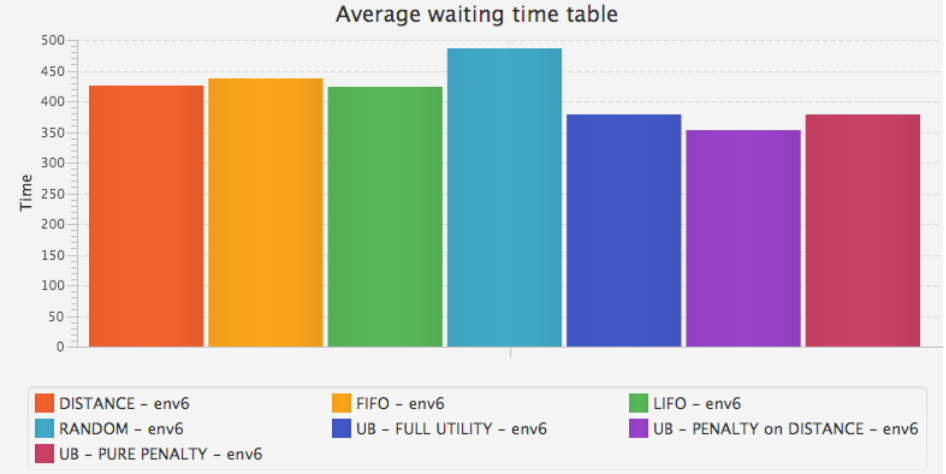

Figura 3.31: Grafico comparativo dei tempi d'attesa per l'*env6*.

tempi medi a seconda che le strategie facciano o non facciano uso dell'utilità.

3.4.7 Le migliori due strategie

Per comparare le varie strategie in env diversi abbiamo sfruttato le singole classifiche assegnando 3 punti al vincitore, 2 al secondo classificato e 1 al terzo classificato. Le altre posizioni guadagnano invece 0 punti. Nel caso di

parità sono stati assegnati i punti corrispondenti alla posizione, indistintamente dal fatto che altre strategie abbiano totalizzato lo stesso valore di penalità.

Strategia	Punti
Penalty on Distance	6
Full Utility	6
Pure Penalty	3
FIFO	3
Random	2
LIFO	0
Distance	0

Le due strategie che quindi testeremo sugli env rimanenti sono quindi PoD e FU.

3.5 Fase 2: test sui rimanenti quattro env

3.5.1 Sperimentazione sull'env3

Penalty on Distance

La strategia PoD fa parte delle strategie basate sull'utilità. Si comporta come la PP ma il valore di penalità è diviso per la distanza (calcolata come nella strategia Distance).

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.32.

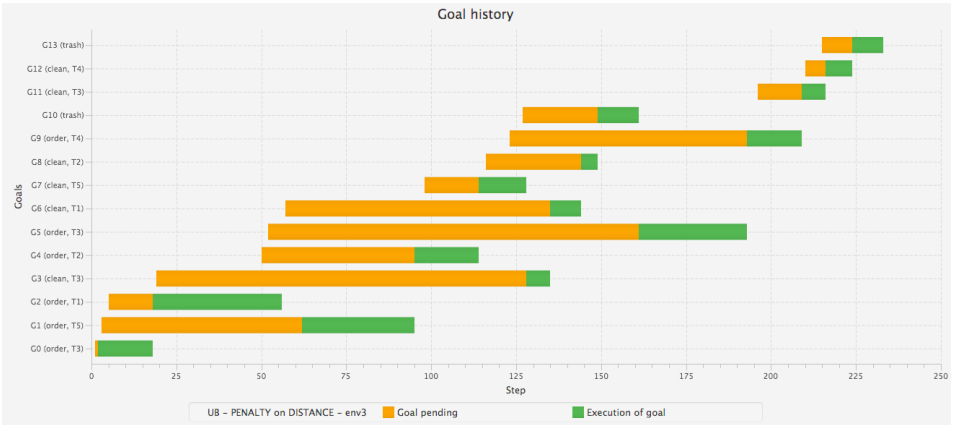

Figura 3.32: Scheduling dei goal con strategia PoD.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	505 (di cui 71,9% copiati)
Ripianificazioni	1
Tempo di esecuzione	2,733 secondi
Penalità totale	13.508
Fatti generati	135.790
Tempo di attesa medio	217,500 unità

Full Utility

La strategia FU fa parte delle strategie basate sull'utilità. I valori di distanza appaiono anche al numeratore per pesare la penalità assegnate ad ogni goal.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.33.

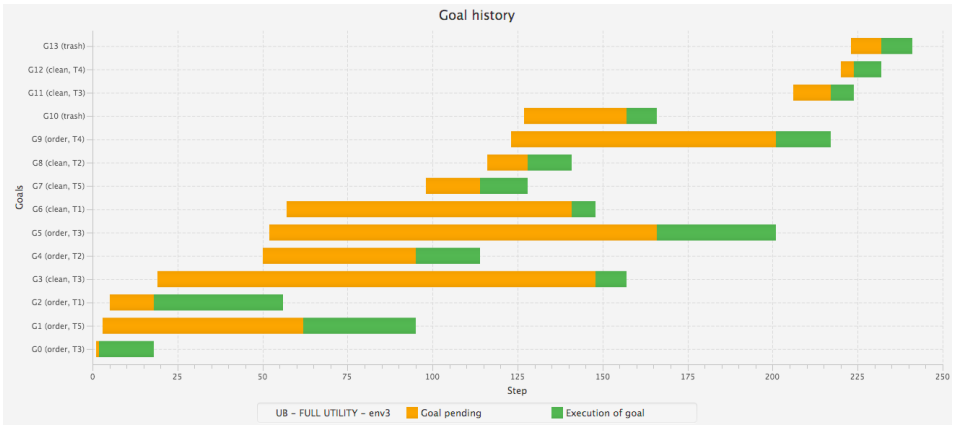

Figura 3.33: Scheduling dei goal con strategia FU.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	505 (di cui 71,9% copiati)
Ripianificazioni	1
Tempo di esecuzione	2,675 secondi
Penalità totale	13.343
Fatti generati	135.626
Tempo di attesa medio	221,5 unità

3.5.2 Comparazione dei risultati sull'env3

Dati statistici. I dati statistici ottenuti in merito all'*env3* sono i seguenti:

Strategia	Penalità	Distanza dalla media
Full Utility	**13.343**	-221
Penalty on Distance	13.508	-56

FU si rivela essere la strategia vincente con poca distanza da PoD.

Andamento della penalità. Il grafico comparativo delle penalità per quanto concerne l'*env3* è il seguente:

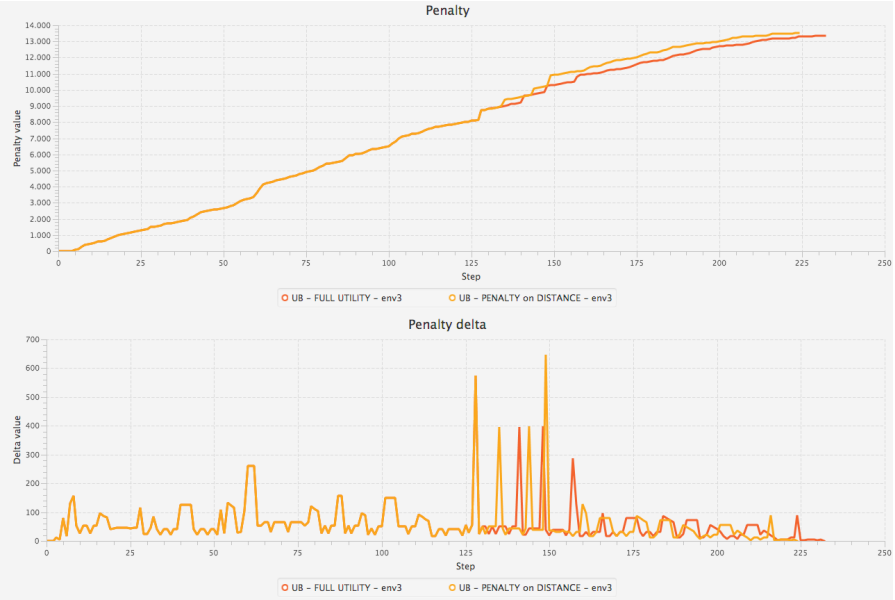

Figura 3.34: Grafico comparativo delle penalità.

3.5.3 Sperimentazione sull'env5

Penalty on Distance

La strategia PoD fa parte delle strategie basate sull'utilità. Si comporta come la PP ma il valore di penalità è diviso per la distanza (calcolata come nella strategia Distance).

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.35.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Figura 3.35: Scheduling dei goal con strategia PoD.

Piani totali	95 (di cui 40% copiati)
Ripianificazioni	3
Tempo di esecuzione	1,976 secondi
Penalità totale	6.658
Fatti generati	91.870
Tempo di attesa medio	200,500 unità

Full Utility

La strategia FU fa parte delle strategie basate sull'utilità. I valori di distanza appaiono anche al numeratore per pesare la penalità assegnate ad ogni goal.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.36.

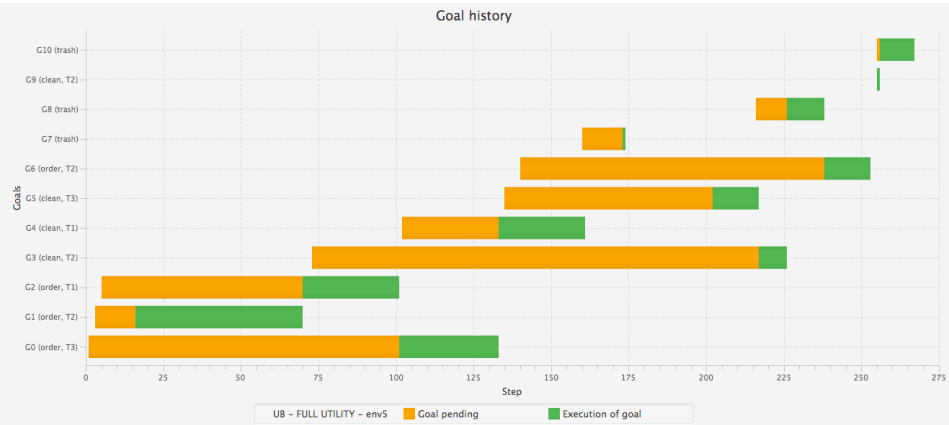

Figura 3.36: Scheduling dei goal con strategia FU.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	118 (di cui 32,2% copiati)
Ripianificazioni	4
Tempo di esecuzione	2,145 secondi
Penalità totale	7.283
Fatti generati	108.894
Tempo di attesa medio	232,75 unità

3.5.4 Comparazione dei risultati sull'env5

Dati statistici. I dati statistici ottenuti in merito all'*env5* sono i seguenti:

Strategia	Penalità	Distanza dalla media
Penalty on Distance	**6.658**	-2.927
Full Utility	7.283	-2.302

PoD è la strategia vincente mentre FU ha un costo decisamente più alto. Andando a studiare ciò che avviene durante l'esecuzione, vediamo che dallo step 175 allo step 200 l'agente è costretto a tornare sui suoi passi per poter permettere ad una persona di attraversare il corridoio (l'*env5* è particolarmente ostico da questo punto di vista). Accade quindi che per 25 step l'agente non debba far altro che evitare questa persona: durante questi 25 step la FU acquisisce 500 penalità circa.

La ragione per cui questa situazione non si presenta anche in PoD è dettata dal fatto che intorno allo step 150 quest'ultima strategia scelga differentemente e quindi non si trovi a dover schivare alcuna persona. Questo risultato ci ricorda quindi di come una scelta ragionata dell'agente possa in realtà portarlo in situazioni che gli faranno acquisire molta più penalità del dovuto e di come il movimento delle persone possa influenzare il punteggio finale.

Andamento della penalità. Il grafico comparativo delle penalità per quanto concerne l'*env5* è mostrato in Figura 3.37.
Abbiamo una ulteriore conferma della situazione descritta in precedenza: l'esecuzione della strategia FU prosegue per più step rispetto a PoD.

3.5.5 Sperimentazione sull'env7

Penalty on Distance

La strategia PoD fa parte delle strategie basate sull'utilità. Si comporta come la PP ma il valore di penalità è diviso per la distanza (calcolata come nella strategia Distance).

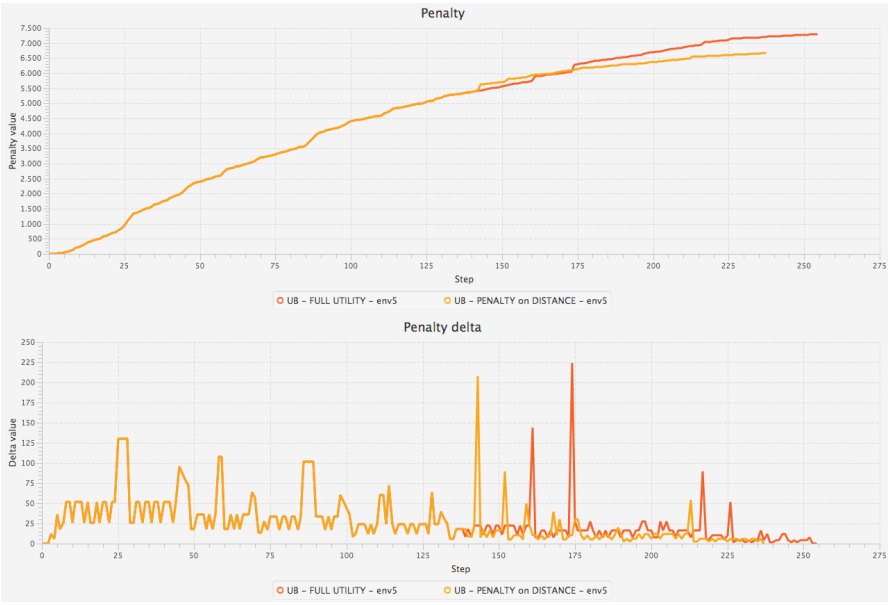

Figura 3.37: Grafico comparativo delle penalità.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.38.

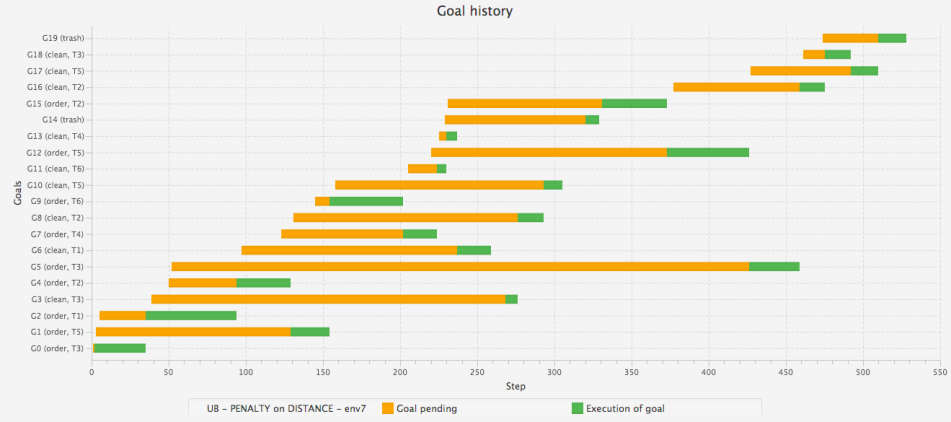

Figura 3.38: Scheduling dei goal con strategia PoD.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	2.586 (di cui 81,6% copiati)
Ripianificazioni	3
Tempo di esecuzione	49,500 secondi
Penalità totale	25.965
Fatti generati	2.012.635
Tempo di attesa medio	361,000 unità

Full Utility

La strategia FU fa parte delle strategie basate sull'utilità. I valori di distanza appaiono anche al numeratore per pesare la penalità assegnate ad ogni goal.

Scheduling dei goal. L'esecuzione della strategia ha portato alla schedulazione dei goal riportata in Figura 3.39.

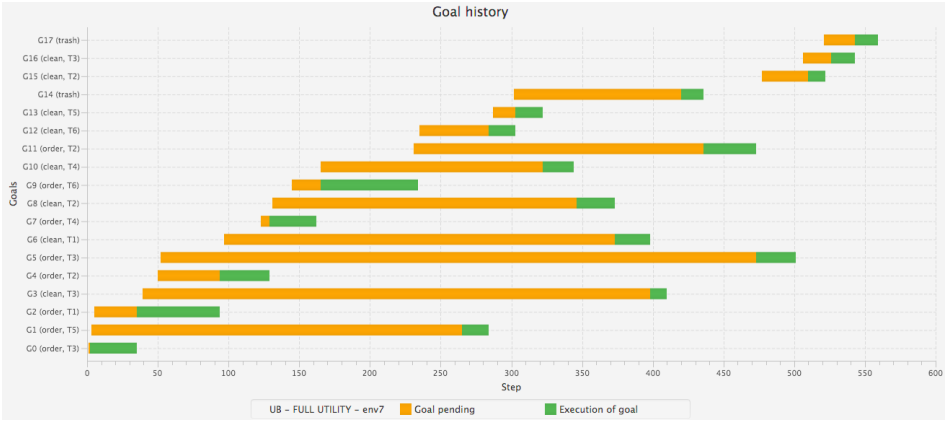

Figura 3.39: Scheduling dei goal con strategia FU.

Dati statistici. L'esecuzione ha prodotto i seguenti dati:

Piani totali	2.148 (di cui 76,2% copiati)
Ripianificazioni	7
Tempo di esecuzione	56,983 secondi
Penalità totale	26.730
Fatti generati	2.499.582
Tempo di attesa medio	391,63 unità

3.5.6 Comparazione dei risultati sull'env7

Dati statistici. I dati statistici ottenuti in merito all'*env7* sono i seguenti:

Strategia	Penalità	Distanza dalla media
Penalty on Distance	**25.965**	-11.865
Full Utility	26.730	-11.100

I punteggi di penalità sono vicini fra loro ma PoD si comporta meglio in termini di penalità e di tempo medio di attesa.

Andamento della penalità. Il grafico comparativo delle penalità per quanto concerne l'*env7* è il seguente:

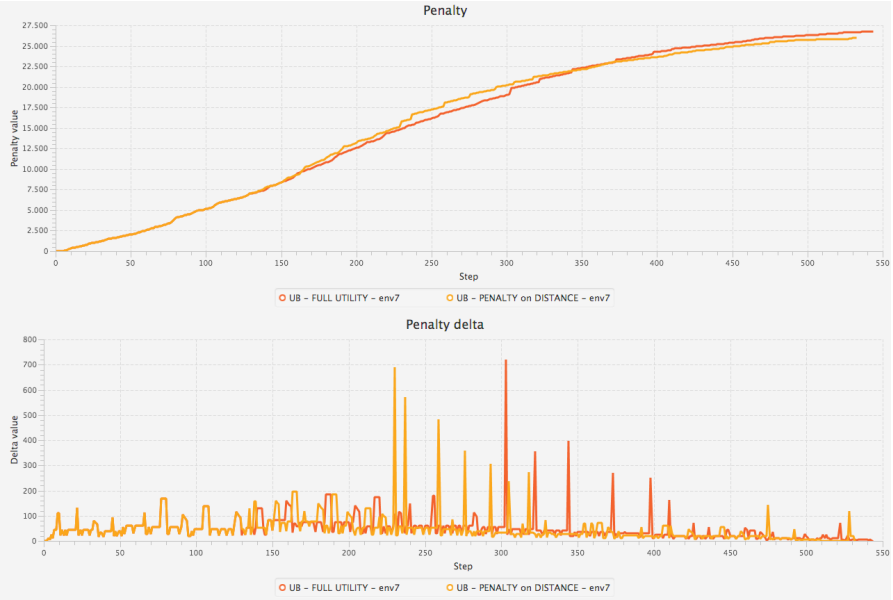

Figura 3.40: Grafico comparativo delle penalità.

3.5.7 La migliore strategia

In due casi su tre è la strategia PoD ad avere la meglio. Si comporta inoltre in tutti e sette i casi meglio della media e quindi possiamo definirla la strategia che si comporta generalmente meglio. L'impiego di sette env rende i nostri test indicativi ma comunque non definitivi: per poter definire PoD una strategia bilanciata avremmo bisogno di molti più test ed environments.

Andremo ora a testare due ulteriori migliorie confrontando i risultati ottenuti prima dell'applicazione di tali migliorie rispetto a quelli ottenuti dopo la loro applicazione.

3.6 Fase 3: ulteriori migliorie

3.6.1 Descrizione delle migliorie

Abbiamo delineato tre ulteriori migliorie che si possono apportare per cercare di diminuire ulteriormente la penalità oppure di ridurre i tempi di esecuzione. Tali migliorie sono:

- Introduzione di tutte le varianti di ordinamento dei `task` di un goal.
- Scelta casuale delle `service-cell` a cui accedere (anziché testare tutte le possibili `service-cell` intorno ad un oggetto).
- In fase di scelta delle `service-cell` a cui accedere, ridurre il numero di celle considerate sfruttando la distanza di Manhattan.

Implementeremo soltanto le prime due migliorie. Ci aspettiamo che la prima aumenti il costo computazionale ma talvolta migliori il punteggio di penalità, mentre la seconda migliori i tempi di esecuzione a discapito della penalità.

3.6.2 Miglioria: ordinamento dei task

Vi sono alcuni ordini nei quali è possibile scegliere di andare prima a caricare il `Food` e poi il `Drink` e viceversa. Fino ad ora abbiamo sempre imposto una scelta mentre con questa miglioria permettiamo all'agente di provare entrambe le versioni e scegliere la migliore.
Prima dell'applicazione della miglioria abbiamo ottenuto questi risultati:

Ambiente	Penalità	Piani totali	Fatti generati	Tempo di esec.
env1	341	47	30.772	1,382
env2	4.529	233	74.763	1,963
env3	13.508	505	135.790	2,733
env4	2.924	52	50.230	1,770
env5	6.658	95	91.870	1,960
env6	28.673	1.617	1.569.802	39,641
env7	25.965	2.586	2.012.635	49,500

In seguito, invece, abbiamo ottenuto i seguenti risultati:

Ambiente	Penalità	Piani totali	Fatti generati	Tempo di esec.
env1	341	74	37.509	1,547
env2	4.521	404	108.122	2,641
env3	11.710	858	191.881	3,708
env4	2.924	52	50.230	1,770
env5	6.658	95	91.870	1,960
env6	27.121	2.824	1.983.243	51,900
env7	25.857	4.946	3.191.000	83,120

Come vediamo *env4* ed *env5* non hanno subito variazioni. Questo è motivato dal fatto che durante la loro esecuzione non esistono piani la cui inversione sia leggitima. il miglioramento dunque non ha sortito alcun effetto. Andando invece a vedere la differenza per quanto riguarda gli altri cinque environment abbiamo:

Ambiente	Penalità	Piani totali	Fatti generati	Tempo di esec.
env1	0	+27	+6.737	+0,165
env2	-8	+171	+33.359	+0,678
env3	-1.798	+303	+56.091	+0,975
env6	-1.552	+1.207	+413.441	+12,259
env7	-108	+2.360	+1.178.365	+33,62

Questa miglioria permette dunque di diminuire la penalità assunta nel 100% dei casi ma con l'ingrandirsi della mappa notiamo anche che i tempi sono di gran lunga superiori sebbene il guadagno in penalità possa non essere così rilevante.

Abbiamo anche notato un interessante comportamento che si evince dal grafico delle penalità relativo all'*env1*:

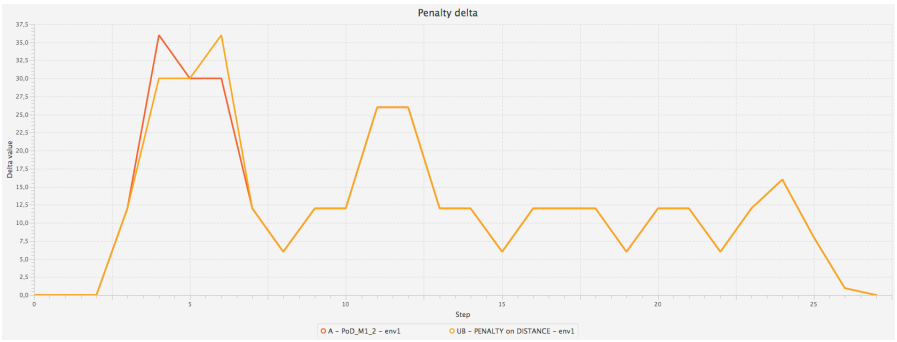

Figura 3.41: Scheduling dei goal con strategia PoD.

Nei primi step notiamo una totale inversione della curva dovuta alla scelta di effettuare prima le `LoadDrink` delle `LoadFood`. Questa situazione si verifica proprio nell'`env1` poiché nella prima mappa i due dispenser sono simmetrici rispetto alla posizione dell'agente e quindi i due picchi che vediamo sono generati a valori diversi a causa del costo differenziato della `LoadFood` e della `LoadDrink`.

3.6.3 Miglioria: scelta casuale delle `service-cell`

Come già detto, attualmente l'agente prova tutte le possibili combinazioni di `service-cell` al fine di trovare il piano ottimo. Con questa miglioria l'agente sceglie, per ogni oggetto, la prima `service-cell` disponibile. L'intento è

quindi quello di confrontare i tempi di esecuzione e le penalità ottenute.
Prima dell'applicazione della miglioria abbiamo ottenuto questi risultati:

Ambiente	Penalità	Piani totali	Fatti generati	Tempo di esec.
env1	341	74	30.772	1,382
env2	4.529	233	74.763	1,963
env3	13.508	505	135.790	2,733
env4	2.924	52	50.230	1,770
env5	6.658	95	91.870	1,960
env6	28.673	1.617	1.569.802	39,641
env7	25.965	2.586	2.012.635	49,500

In seguito, invece, abbiamo ottenuto i seguenti risultati:

Ambiente	Penalità	Piani totali	Fatti generati	Tempo di esec.
env1	241	6	19.649	1,183
env2	4.839	14	32.205	1,309
env3	14.255	29	39.327	1,404
env4	2.887	18	43.542	1,408
env5	#	#	#	#
env6	34.401	51	236.901	7,575
env7	40.629	58	236.628	7,149

Analizzando le differenze tra gli altri cinque environment abbiamo:

Ambiente	Penalità	Piani totali	Fatti generati	Tempo di esec.
env1	-100	-68	-11.123	-0,199
env2	+310	-219	-42.558	-0,654
env3	+747	-476	-96.463	-1,329
env4	-37	-34	-6.688	-0,362
env5	#	#	#	#
env6	+5.728	-1.566	-1.332.901	-32,066
env7	+14.664	-2.528	-1.776.007	-42,451

I dati ottenuti sono interessanti. Nel caso dell'`env1` abbiamo addirittura una importante diminuzione della penalità: il motivo per cui questo accade è che il piano trovato dall'agente evita la persona che invece viene incontrata utilizzando il piano ottimo. Per poter quindi fare un confronto legittimo abbiamo rimosso la persona in una esecuzione con il calcolo di tutti i piani e abbiamo ottenuto una penalità di 205 (che è minore di 241 ottenuto utilizzando un piano casuale).

Un'altra situazione interessante si può riscontrare nell'`env5`: l'agente rimane bloccato in un angolo da una persona come illustrato in Figura 3.42.
Questo tipo di evento può capitare anche quando i piani vengono scelti in maniera non casuale. Ci aspettiamo comunque un environment collaborativo e

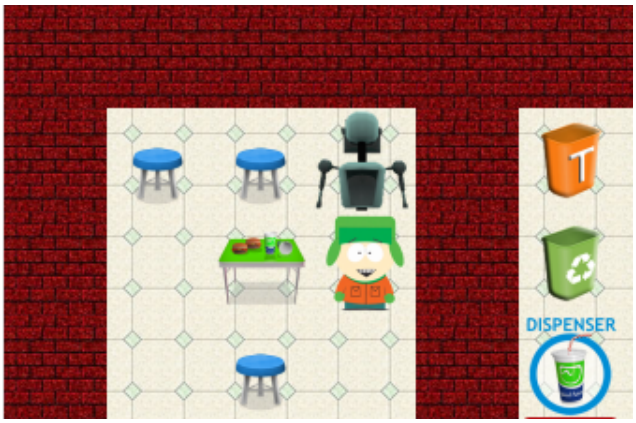

Figura 3.42: Una persona chiude l'agente in un angolo.

dunque possiamo catalogare questo avvenimento come sfortunato.

La particolarità più interessante che abbiamo trovato riguarda l'*env4* che diminuisce il suo punteggio di penalità. Andando a studiare gli eventi delle due esecuzioni (Figura 3.43) capiamo il perché di questo risultato. Entrambe le

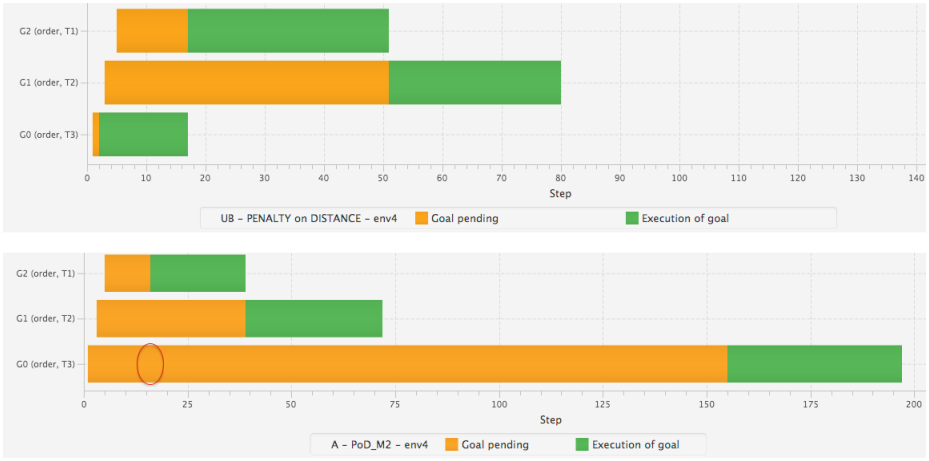

Figura 3.43: Confronto fra le storie relative all'*env4* prima e dopo l'applicazione della miglioria. La zona cerchiata in rosso indica l'abort di un goal.

storie scelgono per primo l'ordine di T3, ma l'esecuzione che fa uso della miglioria incontra una persona: questo fa abortire il goal attualmente scelto e permette all'agente di riconsiderare tutti i goal e sceglierne uno nuovo. Negli step già trascorsi dell'esecuzione è intanto arrivato l'ordine di T2 che viene scelto come nuovo goal e quindi modifica l'intera esecuzione.

Questa peculiare situazione ci fa intuire la potenza di un agente open-minded, in grado di modificare il goal scelto ad ogni step.

In generale notiamo comunque una grande diminuzione (dell'80% circa) dei tempi di esecuzione negli ambienti in cui la pianificazione è più costosa quali *env6* ed *env7*. Per quanto riguarda gli altri env non notiamo una grande differenza in termini di tempo di esecuzione ma un aumento significativo delle penalità.
Deduciamo dunque che in ambienti di grandi dimensioni può essere interessante limitare il numero delle `service-cell` sulle quali si fa pianificazione (eventualmente introducendo una euristica) oppure definire un limite di varianti di piani calcolati per ogni pianificazione.

Capitolo 4

Riflessioni finali e sviluppi futuri

L'agente che abbiamo costruito è predisposto all'espansione di alcune parti e alla miglioria di altre. Scopo di questa sezione è l'elenco delle possibili migliorie effettuabili.

4.1 Diminuire la penalità

Esistono alcune tecniche che si potrebbero introdurre e testare al fine di diminuire la penalità assunta:

- Qualora non ci siano goal pendenti, l'agente potrebbe avvicinarci ad un food dispenser, un drink dispenser o un determinato tavolo a seconda della storia pregressa.

- Alcuni goal potrebbero essere fusi al fine di ottenerne uno solo. Possiamo pensare a due ordini tali che la somma delle quantità richieste sia minore di quanto l'agente può trasportare.

4.2 Diminuire la computazione

I tempi di computazione che abbiamo rilevato non sono esageratamente ampi: anche su grandi ambienti trovare il piano ottimo non costa più di 30 secondi. Riassumendo quanto detto in Sezione 1.3.3, se si desiderasse avere un agente open-minded si potrebbe:

- Fare pruning delle service cell dalle quali ottenere varie versioni di piano sfruttando una euristica.

- Generalizzare le planned-action e permetterne la copia anche da goal pregressi o da piani pre-cablati.

Si dovrebbe inoltre aggiungere una funzione di *reconsider* sufficientemente veloce in mondo da rendere il vantaggio di avere un agente open-minded non troppo costoso.

Si noti inoltre che, attualmente, se un task qualsiasi facente parte di un goal risulta inattuabile, tutto il goal risulta inattuabile. Ma a meno che il task fallito sia proprio il primo, potremmo comunque portare il piano in esecuzione sperando che i task successivi, con il passare del tempo, risultino fattibili e quindi pianificabili online.

4.3 Migliorare le strategie

Vi sono alcuni fattori che non abbiamo considerato in fase di sviluppo delle strategie perché, come accennato nella Sezione 2.1.1, si tratta di migliorie il cui successo è incerto: con maggiore tempo a disposizione si sarebbe potuto effettuare un numero sufficiente di test tale da valutarne l'effettiva efficacia.

Oltre alla miglioria già descritta relativa al modulo `EURISTIC`, potrebbe essere vantaggioso non rispondere ad un ordine se non abbiamo tempo sufficiente per espletarlo. La difficoltà sta nell'approssimare il calcolo di quanto tempo ci vorrà per espletare l'ordine. Supponendo comunque di avere una storia di eventi tali per cui l'agente è in grado di risolvere ogni ordine in tempo, il problema non dovrebbe sussitere.