Nicola Melloni Matr. 794324

RELAZIONE DEL PROGETTO DI INTELLIGENZA ARTIFICIALE: Agente aspirapolvere a conoscenza incompleta

1. Introduzione

Il progetto consiste nell'implementazione di un robot agente aspirapolvere in grado di muoversi in uno spazio e prendere decisioni complesse al fine del raggiungimento dell'obiettivo, anche in circostanze di conoscenza incompleta del mondo.

2. Descrizione del problema

L' obiettivo dell'agente è l'elaborazione di un piano efficiente volto alla pulizia dello spazio in cui è inserito, nel rispetto dei vincoli dati dall' ambiente (la presenza di ostacoli invalicabili) o dalla struttura interna dell'agente stesso (massima capacità di trasporto, massima durata della batteria).

E' possibile avviare una simulazione del comportamento dell'agente al variare di questi fattori attraverso l'esecuzione della procedura **start** nel file **main.pl:**

```
:- start(SpazioDaPulire,DurataBatteria,CapacitaDiTrasporto)
```

All' avvio verranno aperte due finestre rappresentanti lo stato dell'agente nel corso della pulizia:

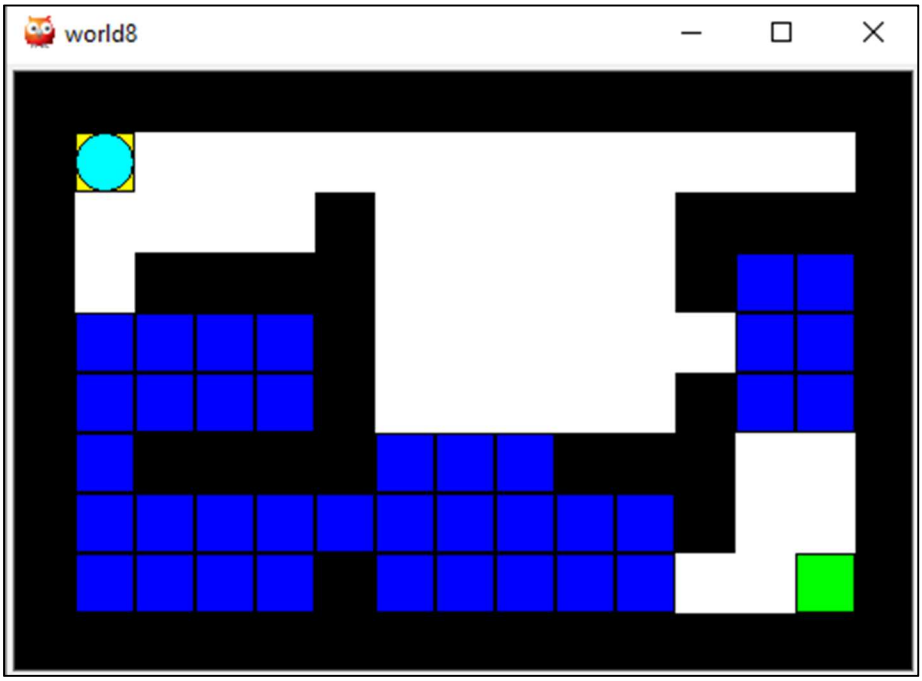


FIGURA 1. AMBIENTE IN CUI L'AGENTE SI MUOVE

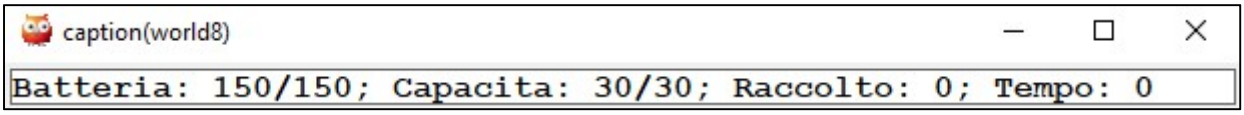


FIGURA 2. PANNELLO DI STATO INTERNO DELL' AGENTE

Lo spazio è rappresentato all' interno dell'applicazione da una matrice rettangolare, in cui ogni cella può essere libera, occupata da uno ostacolo (caselle in nero) o essere marcata come sporca (caselle in blu).

L'agente (cerchio azzurro) mira al raggiungimento di tutte le caselle al fine di raccoglierne lo sporco, e a depositare lo sporco raccolto nel cestino (casella verde) utilizzando al meglio la propria capienza interna, il cui stato è riportato nel pannello.

Al termine della pulizia, l'agente deve tornare alla propria base (casella in giallo) cui è costretto a recarsi per elaborare le prossime mosse e ricaricare la batteria nel caso questa risulti in fase di esecuzione insufficiente al compimento del lavoro.

Durante l'esecuzione, l'aspirapolvere potrà ritrovarsi di fronte all'imprevisto per cui una cella è particolarmente sporca (rappresentata in figura sottostante dalla casella rossa), e rielaborare di conseguenza il proprio piano fino alla completa pulizia della cella, cosa che può richiedere l'esecuzione di molteplici operazioni per l'agente.

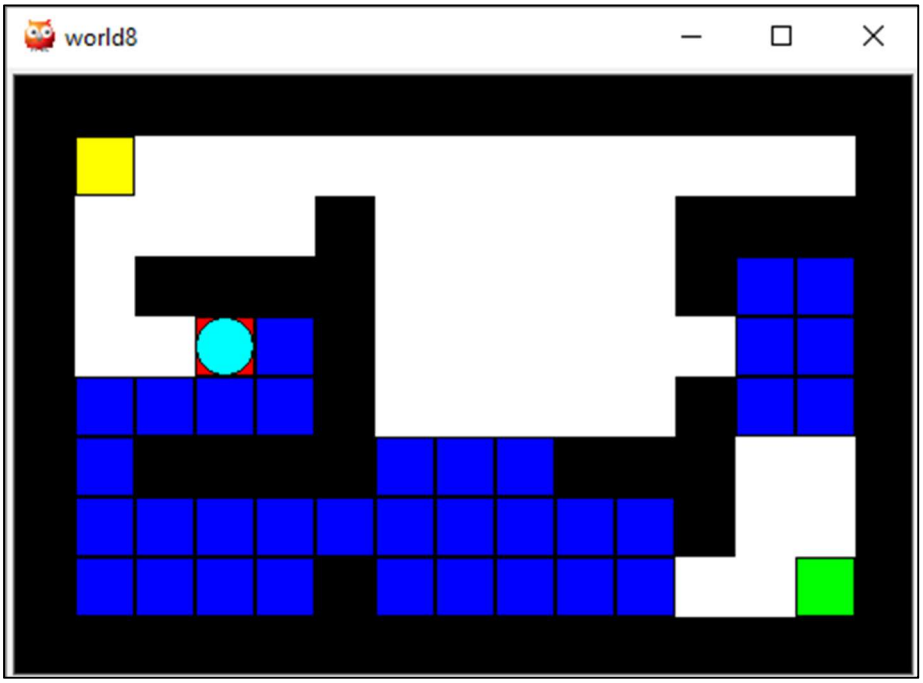


FIGURA 3. REPERIMENTO DI SPORCO IMPREVISTO IN FASE DI ESECUZIONE

Lo spostamento dell'agente di una casella, la pulizia di una casella (blu o rossa) e il deposito dello sporco nel cestino conseguiranno ognuna al consumo di una unità di batteria; la raccolta di sporco (blu o rosso) conseguirà analogamente alla riduzione della capacità di trasporto dell'agente di una unità. Il ritorno alla base o al cestino consegue rispettivamente al ripristino della batteria e capacità di trasporto iniziali.

Obiettivo finale dell'agente è il raggiungimento della totale pulizia dello spazio, minimizzando il numero di volte per cui è costretto a tornare a ricaricare la batteria.

3. Struttura del progetto

Il progetto consiste dei file a seguito descritti:

- **main.pl:** entry point dell'applicazione, che implementa le procedure di settaggio dell'ambiente e contiene i templates dei mondi che è possibile utilizzare per la simulazione.

- **lib:** Libreria contente i moduli utilizzati per la soluzione del problema. L' applicazione si avvale dei seguenti moduli della libreria:

    - **search.pl:** Libreria per la ricerca di una soluzione all' interno dello spazio degli stati. A* con potatura dei chiusi è l'algoritmo impiegato dall' agente per la risoluzione del problema.
    - **strips.pl, strips_planner.pl**: Libreria utilizzata per modellare il comportamento dell'agente secondo il paradigma STRIPS.

- **graphics.pl:** Modulo per la rappresentazione grafica della mappa e del pannello di stato.

- **worlds.pl:** Modulo utilizzato per il caricamento e la generazione di mondi da stringhe di caratteri. Sono inoltre qui contenute funzioni ausiliarie impiegate per la descrizione della geometria del mondo utili a modellare i movimenti dell'agente.

- **two_level_planner.pl:** Modulo contenente l'implementazione aperta di un agente STRIPS in grado di elaborare piani a due diversi livelli di profondità.

- **raccoglitore_gerarchico.pl:** File contenente le dichiarazioni delle azioni eseguibili dall' agente attraverso i predicati add/del esposti dal modulo two_level_planner.pl e le euristiche impiegate nella ricerca di una soluzione.

- **raccoglitore_gerarchico_h0:** File contenente l' implementazione dell' agente ottimo utlizzato per scopi di test

- **agente_ci.pl:** File contenente l'implementazione delle azioni concerete dell'agente e i metodi per l'esecuzione del piano calcolato.

- **testing.pl:** File contenente le procedure per il testing e benchmarking delle performance dell'agente

4. Descrizione dell'agente

Al fine di risolvere il problema in maniera efficiente e nel rispetto dei requisiti, l'implementazione dell'agente include tecniche che, pur compromettendo l'individuazione di una soluzione ottima, apportano grandi benefici ai tempi di esecuzione, altrimenti proibitivi per la natura del problema.

Oltre a ciò, non essendo completa la conoscenza del mondo (l'agente non sa quali sono le caselle più sporche, né sa quanta batteria richiede la loro pulizia), il ricorso ai tradizionali modelli di ottimizzazione su grafo si dimostrerebbe nella pratica insoddisfacente.

Come conseguenza delle precedenti assunzioni, la tecnica di ricerca dell'agente viene modellata alla valorizzazione degli obiettivi descritti dai seguenti paragrafi.

4.1 Implementazione di un'euristica "aggressiva"

Per raggiungere performance accettabili nella soluzione di un problema "difficile" (il problema può infatti considerarsi una variante del problema del Commesso Viaggiatore), l'algoritmo utilizza una euristica che convergere a una soluzione accettabile in tempi molto rapidi.

La tecnica di risoluzione adottata dall' agente è espressa dal seguente blocco di pseudo-codice:

```
if (è possibile pulire dello sporco) {
      raccogli lo sporco secondo l' euristica h1;
} else if (è possibile andare al cestino) {
      vai al cestino;
} else vai a caricare;
```

dove l' euristica $h1$ è una funzione a valori reali indicatrice della tecnica assunta dall' agente nel raccoglimento dello sporco.
La funzione euristica h che implementa tale comportamento è espressa come:

$$h(Stato) = bstteriaResidua + (batteriaMassima + 1) * \\ * (batteriaMassima - sporcoRaccolto + \frac{1}{capacitaMassima + 1} \\ * \left(1 + capacitàConsumata + \frac{batteriaConsumata + h1(Stato)}{batteriaConsumata + h1(Stato) + 1}\right) \\ )$$

Si può notare che per qualsiasi valore assunto da $h1$, l'euristica tenderà a preferire mosse secondo la logica sopra descritta. Nell' implementazione corrente, il valore di $h1$ è posto sempre a zero, di conseguenza l'agente assumerà un comportamento "greedy", e sceglierà di raccogliere lo sporco in modo da minimizzare il consumo di batteria.

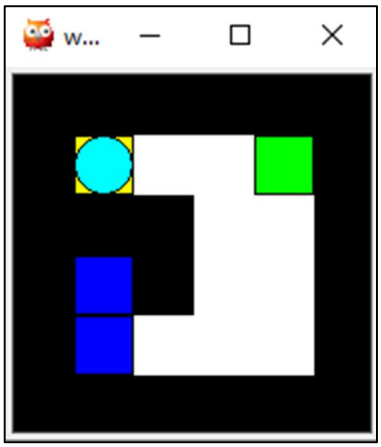
4.2 Riduzione nella dimensione dello spazio degli stati

Per ridurre la dimensione dello spazio degli stati e il dominio della ricerca di una soluzione, l'agente è configurato per escludere dal piano alcune scelte, seppure queste possono poi rivelarsi le migliori, e d'altra parte, a prendere decisioni che si discostano dall' ottimo anche in modo evidente.

Consideriamo ad esempio la seguente regola alla riga 68 del file raccoglitore_gerarchico.pl (in cui P1 rappresenta la posizione dell'agente e P2 una possibile destinazione da pulire):

```
foreach(member(sporco(P),St),(P == P2; not(point_between(P,P1,P2))))
```

La precedente regola impone all' agente di escludere dalle possibili scelte quelle che comprendono altre celle sporche nell' area rettangolare tra P1 e P2. Tale regola corrisponde alla soluzione best first in assenza di ostacoli (che già di per sé è una mossa non garantita come ottima nella totalità del piano), ma si rivela oltremodo sbagliata in alcuni casi in cui la presenza di un ostacolo è di intralcio al tragitto dell'agente, come rappresentato in figura:



Secondo quanto detto sopra, la cella selezionata come mossa migliore è quella superiore, anche se la mossa si rivela evidentemente erronea.

Il risultato di questa imposizione risulta tuttavia in una drastica riduzione del branching factor nell' individuazione di una soluzione.

Inoltre, come descritto al paragrafo successivo, a questo problema viene trovata soluzione attraverso un meccanismo di gestione delle eccezioni e rielaborazione del piano in fase di esecuzione.

4.3 Gestione delle eccezioni e rielaborazione del piano

Secondo quanto detto precedentemente, l'agente deve poter rielaborare il piano nel caso venga trovata una cella particolarmente sporca, o quando il piano di ricerca di un cammino non si dimostri efficiente. Per gestire questo comportamento, è implementato nel file agente_ci.pl un meccanismo di gestione delle eccezioni in fase di esecuzione descritto dalle seguenti regole:

```
1.  % SE MI SPOSTO DA UNA CELLA ANCORA SPORCA E LA CAPACITA' NON E'  A 0, RIELABORO IL PIANO
    esegui_azione_base(Env,va(P1,P2),ok,failed(va(P1,P2))) :-
        capacita(C1),
        C1 \= 0,
        in(P1),
        sporco(P1,_),
```

```
        step(['Trovato sporco imprevisto lungo il cammino. Ricalcolo del piano...\n']).

2.  % SE DEVO PULIRE UNA CELLA SENZA SPORCO, RIELABORO IL PIANO
    esegui_azione_base(_Env,pulisci(P),_,failed(pulisci(P))) :-
        not(sporco(P,_)),
        step(['Pulito lo sporco in ', P, '. Ricalcolo del piano...\n']).
```

Come risultato, l'agente assumerà un comportamento efficiente anche in caso di imprevisti, e, al contempo, compenserà l'atteggiamento fallace indotto dal taglio nello spazio degli stati.

4.4 Cambio di stato nel comportamento dell'agente

Il meccanismo introdotto basato sull' eccezioni compensa parzialmente alle deduzioni errate dovute alle tecniche di riduzione dei tempi di calcolo, tuttavia può comportare che l'agente assuma inattesi.
In particolare, la regola 1 sopra descritta può portare l'agente a entrare in un ciclo infinito qualora sia costretto a tornare a caricare, ma si trovi su una cella sporca e gli sia precluso il movimento.
Per far fronte a ciò, vengono introdotti due diversi stati nell' esecuzione:

- Quando l'agente è in stato di **ok**, questo si comporta secondo le regole finora descritte.

- Quando in fase di **ko**, la regola 1 per la gestione delle eccezioni viene inibita; lo stato normale viene ripristinato solo a seguito della successiva mossa.

L' implementazione di tale comportamento è data dalle regole di decisione imposte all'agente e riportate nel file agente_ci.pl:

```
decidi(failed(_Env, pulisci(_P),_Piano),_,piano(ok)).
decidi(failed(_Env, va(_P1,_P2),_Piano),_,piano(ko)).
decidi(State,_,piano(State)).
decidi(_,_,esecuzione(chiudi)).
```

Come si può notare, al recepimento di un segnale di errore in fase di movimento, il robot entra in fase di ko, dal quale uscirà solo a seguito di un cambio di stato della totalità del sistema, scongiurando così l'eventualità di entrare in un loop infinito.

5. Testing

L'agente finora proposto rispetta i principi di efficienza prestabiliti rispetto ai tempi di calcolo del piano, tuttavia per la sua valutazione complessiva è necessario avere un riscontro anche sulla bontà del piano calcolato.

La tecnica utilizzata per il testing dell'agente è quella di confrontare la sua esecuzione con quella di un altro agente che, seppur meno efficiente, garantisce sempre l'individuazione di una soluzione ottima.

L'agente di test predisposto a tale scopo è descritto nel file 'raccoglitore_gerarchico_h0.pl'. A differenza dell'agente finora considerato, l'agente_gerarchico_h0 calcola direttamente l'intero piano per la pulizia dello spazio individuando quello richiede il minor consumo di batteria, e solo a seguito di ciò avvia la esecuzione.

Il file testing.pl contiene le funzioni impiegate per il confronto nel comportamento e nelle performance dei due due agenti. L'attività di testing può avvenire attraverso l'invocazione dei seguenti comandi :

```
:- test_on, h0_on.
```

Il comando test_on, disabilita l'interazione dell'utente e l'utilizzo di interfaccia grafica nel corso della simulazione, mentre il comando h0_on imposta come modalità per il calcolo del piano quella che ricercata dall'agente di testing. A seguito di ciò, l'invocazione del comando start avvierà l'esecuzione secondo le modalità precedentemente descritte.

5.1 Benchmarking

Una volta attivata la fase di test, è possibile avviare in modalità batch l'esecuzione dei due agenti in diversi scenari di test, per poi confrontare le soluzioni rispetto alle informazioni raccolte.

Attraverso l' invocazione del comando:

```
:- test_all.
```

è' possibile avviare il confronto e visualizzare in forma tabulare i dati riassuntivi di tutte le esecuzioni svolte. Riportiamo qui alcuni risultati ottenuti per le esecuzioni dei due agenti sui mondi world1, world2, world3, world4 al variare della batteria e carica massima:

n	mode	world	batteria	carica	goal	tempo	raccolto
1	normal	world1	10	3	success	21	3
1	h0	world1	10	3	success	17	3
2	normal	world1	10	10	success	21	3
2	h0	world1	10	10	success	17	3
3	normal	world1	100	3	success	11	3
3	h0	world1	100	3	success	11	3
4	normal	world1	100	10	success	11	3
4	h0	world1	100	10	success	11	3
5	normal	world2	20	3	success	37	3
5	h0	world2	20	3	success	31	3
6	normal	world2	20	10	success	37	3

6	h0	world2	20	10	success	31	3
7	normal	world2	100	3	success	21	3
7	h0	world2	100	3	success	21	3
8	normal	world2	100	10	success	21	3
8	h0	world2	100	10	success	21	3
9	normal	world3	20	3	success	34	4
9	h0	world3	20	3	success	30	4
10	normal	world3	20	10	success	28	4
10	h0	world3	20	10	success	28	4
11	normal	world3	100	3	success	26	4
11	h0	world3	100	3	success	26	4
12	normal	world3	100	10	success	20	4
12	h0	world3	100	10	success	20	4
13	normal	world4	20	3	fail	53	5
13	h0	world4	20	3	fail	0	0
14	normal	world4	20	10	fail	37	5
14	h0	world4	20	10	fail	0	0
15	normal	world4	100	3	success	50	6
15	h0	world4	100	3	success	44	6
16	normal	world4	100	10	success	38	6
16	h0	world4	100	10	success	34	6

E’ possibile notare che, per mondi piccoli, il comportamento dell’agente “greedy” sembra essere non di molto inferiore a quello ottimo; si può d’altra parte constatare, nel corso del test, che nel caso dell’agente ottimo le performance nei tempi di calcolo tendono a deteriorarsi assai rapidamente, fino a diventare inaccettabili per istanze del problema poco più grandi.

6. Conclusioni

L’ implementazione data risulta efficace nell’ esecuzione dei casi di test inseriti nel codice, tuttavia il comportamento dell’agente in situazioni diverse, dove ad esempio lo sporco da pulire non è compattato in stanze, ma sparso casualmente nello spazio, è ancora da verificare.

A rimedio del caso appena considerato, è possibile ad esempio dare una definizione più adatta dell’euristica h1, ad esempio tale per cui venga minimizzato il numero di componenti sconnesse nello sporco, o l’area complessiva dell’inviluppo connesso di tali componenti.

Va inoltre notato, che al crescere della batteria nell’ impostazione dei parametri del problema, il tempo di elaborazione del piano cresce esponenzialmente, motivo per cui, in circostanze reali, il ricalcolo del piano potrebbe risultare un’operazione addirittura più lunga del tempo di ricarica della batteria.

Una soluzione a questo problema potrebbe essere quella di vincolare il calcolo del piano a un numero fissato di oggetti raccolti, e quindi suddividere il piano in sottoparti dai tempi di calcolo meno onerosi.