Esempio aspirapolvere EURISTICHE

Come sperimentare

- Dopo aver impostato il progetto su mondi piccoli, dove l'esplosione combinatoria in fase di pianificazione non dispiega ancora i suoi effetti, conviene sperimentare su mondi più grandi.
- Siccome il problema riguarda la pianificazione, la sperimentazione dovrà essere limitata al planner. Nel caso dell'aspirapolvere procederemo come segue
  - NB. Nelle sperimentazioni qui riportate usiamo strips+forward_planner, dove forward_planner usa path_search con strategia A* e potatura chiusi. Nel codice di forward_planner si è posto:

```
:- write('\nstrategia frontiera A*'), set_strategy(astar).
:- write('\npotatura chiusi'), set_strategy(ps(closed)).
%:- write('\npotatura cicli'), set_strategy(ps(cycles)).
```

INIZIAMO CON EURISTICA 0:

```
add_del(pulisci(Q1), St,  [sporco(S,Q2),raccolto(R1)], [sporco(S,Q),raccolto(R)], Q) :-
        member(in(S), St),
        member(sporco(S,Q), St), Q > 0,
        capacita(C), member(raccolto(R),St), R < C,
        Q1 is min(Q, C-R), Q2 is Q-Q1, R1 is R+Q1.
add_del(va(S), St, [in(S)], [in(SPrec)], 1) :-
        member(in(SPrec), St),  comunica(SPrec,S).
add_del(versa_sporco, St,  [raccolto(0)], [raccolto(R)], R) :-
        member(in(balcone), St), member(raccolto(R), St), R > 0.
h(_St,0).

stato_iniziale(St) :-
        setof(sporco(S,Q), sporco(S,Q) , ListSporco), in(Stanza), raccolto(R),
        list_to_ord_set([in(Stanza),raccolto(R)|ListSporco], St).
stato_goal(S) :-
        member(in(balcone),S), member(raccolto(0),S), not((member(sporco(_,Q),S), Q > 0)), !.
piano(esegui_pulizia, Plan, Cost) :-
        plan(stato_iniziale, stato_goal, _Path, _Goal, Cost, Plan).
```

Sperimentiamo **plan(stato_iniziale, stato_goal, _Path, _Goal, Cost, Plan)** *costruendo diversi stati iniziali di test, ad esempio i seguenti:*

```
stato_iniziale(1, S) :-
  consult(mondo1),
  list_to_ord_set([in(balcone),raccolto(0),
                   sporco(a,16), sporco(b,20), sporco(c,15), sporco(d,5)], S).
stato_iniziale(2, S) :-
  consult(mondo1),
  list_to_ord_set([in(balcone),raccolto(0),
                   sporco(a,66), sporco(b,30), sporco(c,35), sporco(d,50)], S).
stato_iniziale(2, S) :-
  consult(mondo2),
  list_to_ord_set([in(balcone),raccolto(0),
                   sporco(a,66), sporco(b,30), sporco(c,35), sporco(d,50), sporco(e,20)], S).
```

dove i mondi 1 e 2 sono riportati nelle slide in appendice. Richiamiamo il pianificatore sui 3 stati iniziali, a partire dal primo, lanciando plan il predicato di testing:

```
test(K, Cost, Plan) :-
        plan(stato_iniziale(K), stato_goal, _Path,_,Plan, Cost).
```

Per valutare i risultati mettiamo a on il modulo di debug e statisiche della libreria search usata dal pianificatore. Il comando è ds_on (ds_off per disabilitarlo)

10 ?- ds_on.
true

11 ?- test(1,A,B).
Livello 0; nuovo livello / RET :
Livello invariato
ERROR: **Out of global stack**

Già con il test 1 si ha un caso non trattabile. Procediamo come segue

A) Vediamo se nella add_del possiamo ridurre il fattore di branching impedendo alcune scelte possibili ma che fanno solo perder tempo

B) Studiamo delle euristiche

A) Nel nostro aspirapolvere abbiamo, correttamente:

```
add_del(va(S), St, [in(S)], [in(SPrec)], 1) :-
        member(in(SPrec), St),  comunica(SPrec,S).
```

Infatti l'aspirapolvere può andare da una stanza ad una comunicante senza alcun vincolo; però ci sembra una perdita di tempo passare per una stanza sporca senza raccogliere nulla. Un codice che evita questa mossa è :

```
add_del(va(S), St, [in(S)], [in(SPrec)], 1) :-
        member(in(SPrec), St), comunica(SPrec,S),
        (   SPrec = balcone    % non deve pulire il balcone
            ;
            % se la stanza è sporca e ha ancora spazio, con not((Q>0,R<C)) impedisco la scelta va(…)
            member(raccolto(R), St), capacita(C), member(sporco(SPrec,Q), St), not((Q>0,R<C))  ).
```

ATTENZIONE, rischio con scelte come questa. *A volte a noi sembra che una scelta sia una perdita di tempo ma in realtà non lo è. In tal caso perdiamo la soluzione ottimale che passa attraverso la scelta da noi esclusa. Nel caso qui sopra discusso, con una sperimentazione random non sono stati trovati contro-esempi in cui la scelta ottimale è scartata dalla restrizione introdotta. Per esserne sicuri bisognerebbe però dimostrare che non vi sono contro-esempi.*

Con la nuova add_del il test 1 termina con successo:

13 ?- test(1,A,B).
Livello 0; nuovo livello / RET :
Livello invariato

nodi espansi: 26806; iterazioni: 45108; profondita' soluzione: 38; costo: 157
b=1.3077731645127928
A = [va(a), pulisci(10), va(balcone), versa_sporco, va(a), pulisci(6), va(balcone), versa_sporco, va(...)|...],
B = 157 .

Però non si riesce a superare il test 2:

14 ?- test(2,A,B).
Livello 0; nuovo livello / RET :
Livello invariato
ERROR: **Out of global stack**

B) Ulteriori vantaggi si ottengono con una euristica. Un esempio di euristica è il seguente:

```
h(St,H) :- capacita(C),
           (  setof(sporco(S,Q), (member(sporco(S,Q),St), Q>0), QQ)  ;  QQ=[]),!,
           sum(QQ,0,H,C).

% calcolo la sommatoria  Σ_{s ∈ stanze} sporco(s) + distanza_balcone(s)*(sporco(s) div capacita + 1)
% dove sporco(s) div capacita + 1 è il numero di viaggi da fare per portar via tutto lo sporco di s
sum([],Sum,Sum,_).
sum([Q|QQ],Sum1,Sum2,C) :-
           sum_stanza(Q,C,Sum1,Sum),
           sum(QQ,Sum,Sum2,C).
sum_stanza(sporco(S,Q), C, Sum1, Sum2) :-
           distanza_balcone(S,D),
           Sum2 is Sum1 + Q + D*(Q div C + 1).
```

Per sperimentare la bontà dell'euristica, conviene prima provare a vedere cosa si guadagna in un test che ha avuto successo, nel nostro caso il test 1. Con euristica 0 avevamo ottenuto:

```
?- test(1,A,B).
Livello 0; nuovo livello / RET :
Livello invariato
nodi espansi: 26806; iterazioni: 45108; profondita' soluzione: 38; costo: 157
b=1.3077731645127928
A = [va(a), pulisci(10), va(balcone), versa_sporco, va(a), pulisci(6), va(balcone), versa_sporco, va(...)|...],
B = 157 .
```

Con la nuova euristica passiamo da fattore effettivo di branching 1.3 a 1.19 (e da 45108 iterazioni a 1816 iterazioni):

```
?- test(1,A,B).
Livello 0; nuovo livello / RET :
Livello invariato
nodi espansi: 935; iterazioni: 1816; profondita' soluzione: 38; costo: 157
b=1.1972345820444201
A = [va(a), pulisci(10), va(balcone), versa_sporco, va(a), pulisci(6), va(balcone), versa_sporco, va(...)|...],
B = 157 .
```

Per il test 2, che andava out of stack, otteniamo:

?- test(2,A,B).
Livello 0; nuovo livello / RET :
Livello invariato
nodi espansi: 19272; iterazioni: 37118; profondita' soluzione: 109; costo: 793
b=1.0947406759240264
A = [va(a), pulisci(10), va(balcone), versa_sporco, va(a), pulisci(10), va(balcone), versa_sporco, va(...)|...],
B = 793 .

Con il test 3 abbiamo tempi di attesa più lunghi. Alla fine otteniamo:

26 ?- test(3,A,B).
Livello 0; nuovo livello / RET :
Livello invariato
nodi espansi: 58120; iterazioni: 113856; profondita' soluzione: 119; costo: 791
b=1.096569973136056
A = [va(a), pulisci(10), va(balcone), versa_sporco, va(a), pulisci(10), va(balcone), versa_sporco, va(...)|...],
B = 791

Un modo per accelerare la ricerca che spesso funziona è moltiplicare l'euristica per una costante > 1, rendendola più «aggressiva»; il problema è che in questo modo l'euristica potrebbe non essere più sottostimata, per cui si perde l'ottimalità. Vediamo cosa succede moltiplicando l'euristica precedente per 4, cioè con il codice seguente:

```
h(St,H)  :-  (   setof(sporco(S,Q), (member(sporco(S,Q),St), Q>0), QQ) ;   QQ=[]),!,
             sum(QQ,0,H1), H is 4*H1.
```

Con questa euristica test hanno risposta in tempi decisamente più rapidi, fate voi i confronti:

```
28 ?- test(1,A,B).
Livello 0; nuovo livello / RET :
Livello invariato
nodi espansi: 85; iterazioni: 150; profondita' soluzione: 41; costo: 161
b=1.1144459189464158
A = [va(a), pulisci(10), va(balcone), versa_sporco, va(a), pulisci(6), va(d), pulisci(4), va(...)|...],
B = 161 .
```

?- test(2,A,B).
Livello 0; nuovo livello / RET :
Livello invariato
nodi espansi: 235; iterazioni: 434; profondita' soluzione: 109; costo: 793
b=1.051363549743581
A = [va(a), pulisci(10), va(balcone), versa_sporco, va(a), pulisci(10), va(balcone), versa_sporco, va(...)|...],
B = 793 .

?- test(3,A,B).
Livello 0; nuovo livello / RET :
Livello invariato
nodi espansi: 292; iterazioni: 546; profondita' soluzione: 119; costo: 791
b=1.048859911183307
A = [va(a), pulisci(10), va(balcone), versa_sporco, va(a), pulisci(10), va(balcone), versa_sporco, va(...)|...],
B = 791

Come si vede in tutti e 3 i test si è avuto un notevole miglioramento nei tempi di risposta, ma la risposta ottenuta nel test 1 non è ottimale; quella trovata con l'euristica precedente aveva costo 157, mentre quella trovata ora ha costo non ottimale 161>157.
Invece per i test 2 e 3 la nuova euristica ha dato le soluzioni ottimali. Si tratterebbe di vedere sperimentalmente in quanti casi la risposta non è ottimale e di quanto si discosta dalla ottimale.

APPENDICE. mondo1.pl

```
stanza(a).
stanza(b).
stanza(c).
stanza(d).

porta(balcone,a).
porta(a,b).
porta(b,c).
porta(c,d).
porta(d,a).

distanza_balcone(a,1).
distanza_balcone(b,2).
distanza_balcone(c,3).
distanza_balcone(d,2).

comunica(S1,S2) :-
        porta(S1,S2); porta(S2,S1).
```

APPENDICE. mondo2.pl

```
stanza(a).
stanza(b).
stanza(c).
stanza(d).
stanza(e)

porta(balcone,a).
porta(a,b).
porta(b,c).
porta(c,d).
porta(d,a).
porta(d,e).

distanza_balcone(a,1).
distanza_balcone(b,2).
distanza_balcone(c,3).
distanza_balcone(d,2).
distanza_balcone(e,3).

comunica(S1,S2) :-
        porta(S1,S2); porta(S2,S1).
```