HSP - Hill-Climbing Reforçado
Laboratório de Inteligência Artificial

Eudênia Xavier Meneses & Silvio do Lago Pereira

5 de maio de 2003

1 A estratégia de busca utilizada

Nesse trabalho, implementamos um planejador baseado em busca heurística no espaço de estados do problema de planejamento, conforme descrito em [1]. Trata-se de um algoritmo *hill-climbing* com estratégia de busca reforçada, ou seja, ao se expandir um nó da árvore de busca, se nenhum dos filhos encontrados tem custo menor do que o custo do nó expandido, uma busca em largura com limite de profundidade igual a 5 é iniciada (*vide* figura 1). Se durante essa busca em largura um nó com custo menor é encontrado, o caminho já percorrido com a busca *hill-climbing* é estendido com o caminho encontrado pela busca em largura; caso contrário, a busca *hill-climbing* termina com fracasso.

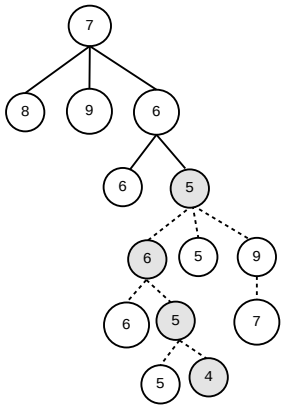

Figura 1: Nessa figura, as linhas contínuas indicam os nós examinados pelo algoritmo *hill-climbing*, que escolhe sempre o primeiro filho que tem custo menor que seu pai. Quando um tal filho não é encontrado, iniciamoss uma busca em largura, indicada pelas linhas pontilhadas, que termina assim que encontramos o primeiro nó com custo menor do que o custo daquele nó que estava sendo expandido pelo algoritmo *hill-climbing*. Caso um tal nó não seja encontrado até, no máximo uma profundidade igual a 5, a busca termina com fracasso.

A seguir, apresentamos uma implementação desse algoritmo PROLOG:

```
% Hill-climbing search.

  hill_climbing(_, _, 0, Plan, Plan) :- !.

  hill_climbing(H, State, Cost, SubPlan, Plan) :-
    extend(H, State,Cost,Path,NextState,NextCost),
    append(SubPlan,Path,NewSubPlan),
    hill_climbing(H, NextState, NextCost, NewSubPlan, Plan).

% SubPlan extension by best-first search.

  extend(H, State, Cost, [Action], NextState, NCost) :-
    succ_state_hc(H, State, Action, NextState, NCost),
    NCost<Cost, !.

  extend(H, State,Cost,SubPlan,NextState,NextCost) :-
    bfs(10, H, Cost,[Cost:State:[]],SubPlan,NextState,NextCost).

% Best-first search.

  bfs(_, _, Cost,[C:NextState:P|_],Plan,NextState,C):-
    Cost>C,
    reverse(P, Plan), !.

  bfs(N, H, Cost,[_:S:P|Rest],Plan,NextState,NextCost):-
    N>0, NN is N-1,
    findall(C1:N1:[A|P], succ_state(H, S,A,N1,C1),Succ),
    append(Rest,Succ,List),
    sort(List,Queue),
    bfs(NN,H, Cost,Queue,Plan,NextState,NextCost).

% Find successors states.

  succ_state(H, State, Action, NextState, Cost) :-
    executable(Action,State),
    update(State,Action,NextState),
    ht_put(NextState),
    cost(H, NextState,Cost).

  succ_state_hc(H, State, Action, NextState, Cost) :-
    executable(Action,State),
    update(State,Action,NextState),
    cost(H, NextState,Cost).

% Find an action that is executable in the given state.

  executable(Action, State) :-
    oper(Action, Pre, _, _),
    subset(Pre, State).

% Update a state with one given action.

  update(State, Action, NextState) :-
    oper(Action, _, Add, Del),
    union(State, Add, S),
    subtract(S, Del, NextState).
```

2 Funções heurísticas

Implementamos duas funções heurísticas: H_{max} e H_{add}. A primeira delas (H_{max}) é admissível mas, conforme observamos com os testes realizados, não é muito informativa. Por outro lado, a heurística H_{add}, que é não-admissível, parece ser bem mais informativa.

2.1 Cálculo das funções heurísticas

Seja $oper(Act, Pre, Add, Del)$ um operador e a um átomo adicionado por esse operador, ou seja, $a \in Add$. A diferença básica entre as duas funções heurísticas é que, enquanto H_{max} toma como custo de a o valor $max\{p : p \in Pre\} + 1$, H_{add} toma o valor $add\{p : p \in Pre\} + 1$. Claramente, H_{max} é admissível. Para ver que o mesmo não acontece com H_{add}, basta considerar um domínio em que um mesmo operador adicione vários átomos do estado meta.

Na verdade, as funções heurísticas empregadas no planejador que implementamos são baseadas numa versão "relaxada" do problema de planejamento, sendo resolvido, em que as listas de efeitos negativos dos operadores aplicados são ignoradas. Para tanto, partindo de um estado corrente S, fazemos uma projeção, usando todo operador aplicável, até atingir um ponto-fixo em que nenhum novo átomo pode ser atingido, nem o custo de um deles pode ser alterado para um valor mais baixo. Nesse ponto, o custo do estado S é calculado tomando-se o máximo ou a soma, conforme a heurística adotada, dos custos dos átomos do estado meta.

2.2 Implementação das funções heurísticas

O procedimento de cálculo das funções heurísticas H_{max} e H_{add}, implementado em PROLOG é apresentado a seguir:

```
calculate_costs(_, C,C,C).
calculate_costs(H, CurrentCosts, _, FinalCosts):-
  extract_state(CurrentCosts, State),
  findall(Action, executable(Action, State), AcList),
  aux_calculate(H, AcList, CurrentCosts, NewCosts),
  calculate_costs(H, NewCosts, CurrentCosts, FinalCosts).

aux_calculate(_, [],C, C).
aux_calculate(max, [Action|As], CCosts, NCosts):-
  oper(Action, Prec, Add, _),
  maxcosts(Prec,CCosts, 0, Max),
  apply_cost(Add, CCosts, Max, NewCosts),
  aux_calculate(max, As, NewCosts, NCosts).

aux_calculate(add, [Action|As], CCosts, NCosts):-
  oper(Action, Prec, Add, _),
  addcosts(Prec,CCosts, 0, Max),
  apply_cost(Add, CCosts, Max, NewCosts),
  aux_calculate(add, As, NewCosts, NCosts).

extract_state([],[]).
```

```
extract_state([(Atom, _)|Rs], [Atom|Ts]):-
  extract_state(Rs,Ts).

maxcosts([],_, Max, NMax):- NMax is Max +1.
maxcosts([Atom|Rs], Costs, MaxInt, Max):-
  member((Atom, Cost), Costs),
  (Cost>MaxInt
   -> maxcosts(Rs,Costs, Cost, Max)
   ;  maxcosts(Rs,Costs, MaxInt, Max)).

addcosts([],_, Add, FAdd):- FAdd is Add +1.
addcosts([Atom|Rs], Costs, AddInt, Add):-
  member((Atom, Cost), Costs),
  NAdd is AddInt+Cost,
  addcosts(Rs,Costs, NAdd, Add).

apply_cost([],Res, _,Res).
apply_cost([Atom|As], CurrentCosts, NewCost, Res):-
  (append(L1,[(Atom, Cost)|L2],CurrentCosts)
   -> ((Cost>NewCost)
        -> append(L1,[(Atom,NewCost)|L2], NewList)
        ;  NewList=CurrentCosts)
   ;  (NewList=[(Atom, NewCost)|CurrentCosts])),
  apply_cost(As, NewList, NewCost, Res).

gmax(State, CurrentState, FCost):-
  apply_cost(CurrentState, [], 0, InitialCosts),
  calculate_costs(max, InitialCosts, [], FinalCosts),
  max_heurist(State, FinalCosts, 0, FCost),!.

max_heurist([],_, C, C).
max_heurist([Atom|As], Costs, IntCost, FCost):-
  (member((Atom, Cost), Costs)
   -> (Cost>IntCost
       -> max_heurist(As, Costs, Cost, FCost)
       ;  max_heurist(As, Costs, IntCost, FCost))
   ;  FCost=99999999999999).

gadd(GoalState, CurrentState, FCost):-
  apply_cost(CurrentState, [], 0, InitialCosts),
  calculate_costs(add, InitialCosts, [], FinalCosts),
  sum_heurist(GoalState, FinalCosts, 0, FCost),!.

sum_heurist([],_, C, C).
sum_heurist([Atom|As], Costs, IntCost, FCost):-
  (member((Atom, Cost), Costs)
   -> (NewCost is Cost+IntCost,
       sum_heurist(As, Costs, NewCost, FCost))
   ;  FCost=99999999999999).
```

3 Conversão de esquemas em operadores

As descrições dos domínios em PDDL foram convertidas em descrições equivalentes em Prolog. Então, escrevemos um programa, também em Prolog, para converter esses esquemas de operadores em operadores proposicionais. O código

do programa de conversão utilizado é apresentado a seguir:

```
% convertion driver

  stop(InputFile, OutputFile) :-
    (exists_file(InputFile)
     -> (consult(InputFile),
         tell(OutputFile),
         domain(Domain),
         oper(act: Act, pre: Pre, add: Add, del: Del),
         expand(Domain, oper(act:Act, pre:Pre, add:Add, del:Del), Instances),
         filter(Instances, [], Operators),
         writeops(Operators),
         fail ; told)
     ;  format('~nFile not found: ~w~n',InputFile)).


% schema expansion

  expand(Domain, Schema, Instances) :-
    Schema = oper(act: Act, pre: Pre, add: Add, del: Del),
    Act =.. [_|Args],
    findall(Schema, instance(Args,Domain,Act,Pre,Add,Del), Instances).


% schema instanciation

  instance([],_,_,_,_,_).

  instance([Arg|Args],Domain,Act,Pre,Add,Del) :-
    (ground(Arg)
     -> instance(Args,Domain,Act,Pre,Add,Del)
     ;  (member(Arg,Domain),
         instance(Args,Domain,Act,Pre,Add,Del))).


% instances filtering

  filter([], Instances, Operators) :-
    reverse(Instances, Operators).

  filter([oper(act:Act,pre:Pre,add:Add,del:Del)|Ins],Lst,Operators) :-
    (consist(Pre, [], NewPre)
     -> filter(Ins,[oper(act:Act,pre:NewPre,add:Add,del:Del)|Lst],Operators)
     ;  filter(Ins,Lst,Operators)).


% consistence checking

  consist([],Pre,NewPre) :-
    reverse(Pre,NewPre).

  consist([P|Ps], Pre, NewPre) :-
    (P = (X\=Y)
     -> ((X\=Y)
         -> consist(Ps, Pre, NewPre)
         ; !,fail)
     ;  consist(Ps, [P|Pre], NewPre)).
```

```
% operators formating

  writeops([]) :-
    nl.

  writeops([oper(act:Act, pre:Pre, add:Add, del:Del)|Ops]) :-
   format('~noper(~w,~n',Act),
   format('     ~w,~n', [Pre]),
   format('     ~w,~n', [Add]),
   format('     ~w).~n',[Del]),
   writeops(Ops).
```

4 Domínios de teste

Os domínios empregados para teste do planejador foram:

- *mundo dos blocos*:

```
domain([a,b,c,d,e]).

oper(act: stack(X,Y),
     pre: [X\=Y, ontable(X), clear(X), clear(Y)],
     add: [on(X,Y)],
     del: [ontable(X), clear(Y)]).

oper(act: unstack(X,Y),
     pre: [X\=Y, on(X,Y), clear(X)],
     add: [ontable(X), clear(Y)],
     del: [on(X,Y)]).

oper(act: move(X,Y,Z),
     pre: [X\=Y, X\=Z, Y\=Z, on(X,Y), clear(X), clear(Z)],
     add: [on(X,Z), clear(Y)],
     del: [on(X,Y), clear(Z)]).
```

- *satélites*:

```
domain([satellite(a),
        direction(north), direction(south),
        direction(est), direction(west),
        instrument(x), instrument(y),
        instrument(z), mode(good), mode(bad)]).

oper(act: turn_to(satellite(X), direction(DNew), direction(DPrev)),
     pre: [DNew\=DPrev, pointing(satellite(X), direction(DPrev))],
     add: [pointing(satellite(X), direction(DNew))],
     del: [pointing(satellite(X), direction(DPrev))]).

oper(act: switch_on(instrument(Y), satellite(X)),
     pre: [on_board(instrument(Y), satellite(X)),
           power_avail(satellite(X))],
     add: [power_on(instrument(Y))],
     del: [calibrated(instrument(Y)), power_avail(instrument(Y))]).
```

```
oper(act: switch_off(instrument(Y), satellite(X)),
     pre: [on_board(instrument(Y), satellite(X)),
           power_on(instrument(Y))],
     add: [power_avail(satellite(X))],
     del: [power_on(instrument(Y))]).

oper(act: calibrate(satellite(X), instrument(Y), direction(D)),
     pre: [on_board(instrument(Y), satellite(X)),
           calibration_target(instrument(Y), direction(D)),
           pointing(satellite(X), direction(D)),
           power_on(instrument(Y))],
     add: [calibrated(instrument(Y))],
     del: []).

oper(act: take_image(satellite(X), direction(D), instrument(Y), mode(M)),
     pre: [on_board(instrument(Y), satellite(X)),
           calibrated(instrument(Y)),
           pointing(satellite(X), direction(D)),
           supports(instrument(Y),mode(M)),
           power_on(instrument(Y))],
     add: [have_image(direction(D), mode(M))],
     del: []).
```

- *sokoban*:

```
domain([bloco(a),
        posicao(1),posicao(2),posicao(3),posicao(4),
        posicao(5),posicao(6),posicao(7),posicao(8),posicao(9),
        posicao(10),posicao(11),posicao(12),
        posicao(13),posicao(14),posicao(15), posicao(16)]).

oper(act: empurrar(posicao(Y), bloco(X), posicao(A), posicao(N)),
     pre: [Y\=A, A\=N, Y\=N,
           posicaolivre(N),
           posicaosokoban(Y),
           posicaobloco(bloco(X), posicao(A)),
           podeempurrar(Y, A, N)],
     add: [posicaobloco(bloco(X),posicao(N)),
           posicaosokoban(A),
           posicaolivre(Y)],
     del: [posicaobloco(bloco(X), posicao(A)),
           posicaosokoban(posicao(Y)),
           posicaolivre(N)]).

oper(act: andar(posicao(A), posicao(N)),
     pre: [A\=N,
           posicaolivre(N),
           posicaosokoban(A),
           adjacente(A,N)],
     add: [posicaosokoban(N), posicaolivre(A)],
     del: [posicaosokoban(A), posicaolivre(N)]).
```

5 Desempenho dos planejadores

O desempenho dos planejadores, guiado pelas duas heurísticas, foi analisado através dos resultados obtidos com experimentos realizados em três domínios de planejamento: *mundo dos blocos*, *satélites* e *sokoban*. O resultado dessa análise é apresentado a seguir.

5.1 Problemas no mundo dos blocos

O resultados obtidos para problemas no mundo dos blocos são apresentados na figura 2. Como podemos observar, a heurística H_{max} teve um desempenho bem pior do que a heurística H_{add}. Conforme mostra o gráfico da esquerda, o tempo de CPU gasto pela heurística H_{max} cresce muito mais rapidamente do que o tempo gasto pela heurística H_{add}. Além disso, como mostra o gráfico do centro, o espaço de busca explorado pelo planejador guiado por H_{max} é também muito maior do que aquele explorado quando esse mesmo planejador é quiado pela heuritica H_{add}, sendo esse o motivo da heurística H_{max} gastar mais tempo para resolver os problemas. Finalmente, como mostra o gráfico da direita, o tamanho dos planos encontrados é o mesmo, com ambas as heurísticas, para problemas idênticos. Assim, com relação à qualidade dos planos encontrados, as heurísticas se comportaram de maneira equivalente.

Outros testes realizados mostraram que a heurística H_{add}, por explorar um espaço de busca menor do que aquele explorado pela heurística H_{max}, também consegue resolver problemas com maior número de blocos. Por outro lado, como a heurística H_{add} é não-admissível, alguns problemas não puderam ser solucionados com essa heurística.

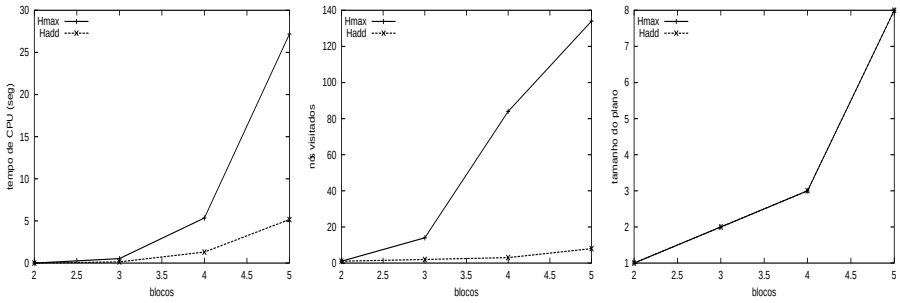

Figura 2: Desempenho para problemas no mundo dos blocos

5.2 Problemas dos satélites

O resultados obtidos para problemas no mundo dos blocos são apresentados na figura 3. Conforme observamos no gráfico da esquerda, para o domínio dos satélites, a heurística H_{max} parece ter um desempenho melhor do a heurística H_{add}. Isso, entretanto, não é verdade. O que acontece é que, para problemas

maiores, a busca guiada por H_{max} termina com fracasso. Então, como a profundidade da busca em largura é limitada, a busca com H_{max} termina antes da busca com H_{add} que, efetivamente, encontra uma solução. Essa afirmação é confirmada pelo gráfico da direita, que mostra que, para o maior problema, o tamanho do plano encontrado por H_{max} é zero. Além disso, como mostra o gráfico do centro, o espaço de busca explorado por H_{max} ainda continua sendo bem maior que o espaço explorado por H_{add}.

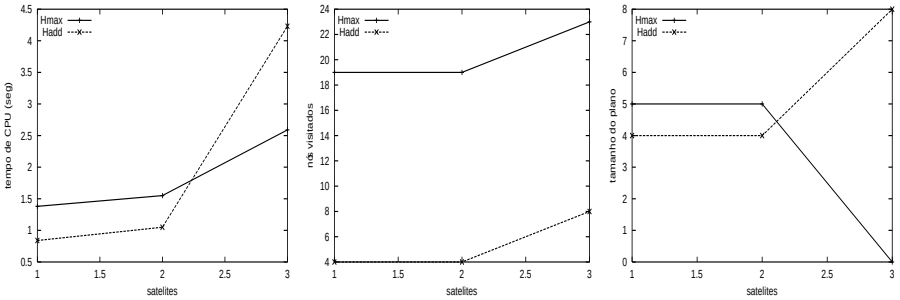

Figura 3: Desempenho para problemas dos satélites

5.3 Problemas do sokoban

O resultados obtidos para problemas no mundo dos blocos são apresentados na figura 4. Observando o gráfico da esquerda, para o problema 2, vemos que H_{add} teve um desempenho melhor; porém, de acordo com o gráfico da direita, o tamanho do plano encontrado é 0. Isso confirma nossa afirmação de que a heurística H_{add} é não-admissível, já que esse mesmo problema para o qual H_{add} não encontrou solução foi corretamente resolvido quando usamos a heurística H_{max}. Nos demais casos, o desempenho de H_{add} ainda parece ser melhor (ou igual) ao desempenho de H_{max}, conforme observamos nos experimentos anteriores.

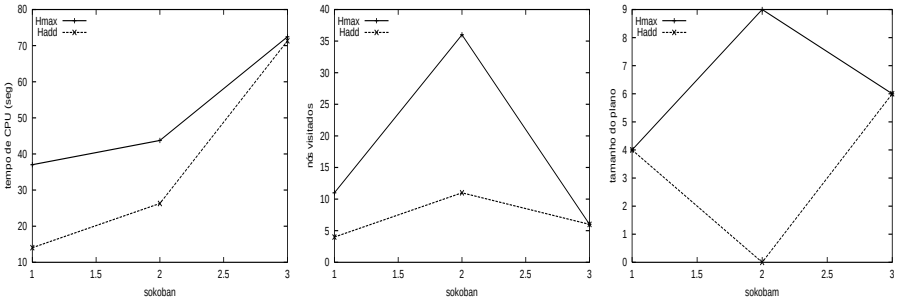

Figura 4: Desempenho para problemas do sokoban

6 Conclusão

Nesse trabalho implementamos um planejador baseado em busca heurística (*hill-climbing* reforçado), sendo que a heurística específica a ser empregada é indicada como um parâmetro de entrada para o programa.

A característica "reforçado", foi implementada por uma busca em largura (*best-first search*) com profundidade limitada. Achamos mais apropriado limitar a profundidade de busca em largura para evitar que a busca *hill-climbing* fosse degenerada a uma busca em largura simples. Com isso, garantimos que os resultados obtidos não são meramente resultados obtidos com busca em largura, mas sim pela combinação das duas estratégias de busca.

Com esse planejador realizamos uma série de experimentos que nos mostram que a heurística H_{add}, embora não-admissível, apresenta um desempenhho melhor que a heurística H_{max}; ou seja, H_{add} é uma heurística mais informativa.

Referências

[1] Bonet, B. & Geffner, H. *Planning as Heuristic Search, Depto. de Computación, Universidad Simón Bolívar, 2000.*