Logic Programming, 16-17

Inês Dutra
DCC-FCUP
ines@dcc.fc.up.pt (room: 1.31)

October 23, 2016

Search Problems

- General problem: given an initial state S_i and a final state S_f, find a path between these two states making sure each transition corresponds to a valid move.
- Example: Hanoi
  - blocks need to be moved one at a time.
  - a block can only be moved if there is nothing on its top.
- To find the solution (sequence of moves), we need to be able to transform our initial blocks state in the final blocks state.

Hanoi example

- Exemplo:

```
+----+          +----+
| C  |          | A  |
+----+          +----+
| A  |   ===>   | B  |
+----+    ??    +----+
| B  |          | C  |
+----+          +----+
/////           /////
```

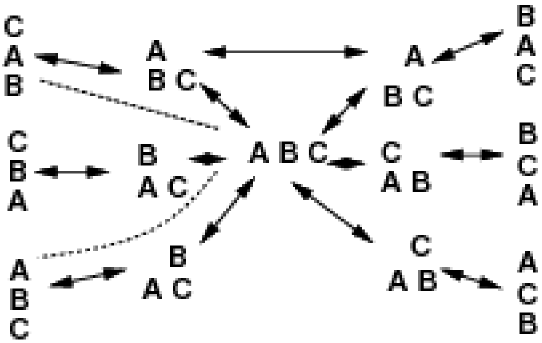
- **what** path to follow to transform config 1 in config 2?
- **how** to find this path?

Hanoi example

- We will explore alternative paths till finding the solution.
- For example, after placing C in the floor, we have the alternatives:
  - place A in the floor, **OR**
  - place A on top of C, **OR**
  - place C on top of A.
- Two types of concepts:
  - situations (states, nodes, configurations).
  - possible movements (actions, operators, transformations) which can transform one state in another state.

Search Problems

Ex:



Search Problems

- Summarizing:
  - state space.
  - initial state.
  - final state (goal).
  - operator that can transform one state in the next.
- Optimization problem: find the path with minimum cost.

Search Problems

- Representation of a state space in Prolog: `s(X,Y)` ou `s(X,Y,C)`, with C a cost of transitioning from state X to Y.
- `s(X,Y)` is true is there is a legal/possible movement from X to Y.
- In the blocks problem (Hanoi), a state can be represented by a list of stacks. Each stack, byt its turn can be represented by a list whose first element is the block on the top of the stack.

Search Problems: defining a transition from X to Y - example

- Initial state: `[[c,a,b],[],[]]`
- Final state: any set of stacks that contains one of the stacks with the blocks ordered:
  - `[[a,b,c],[],[]]`
  - `[[],[a,b,c],[]]`
  - `[[],[],[a,b,c]]`
- Given a state, to find the next state, we use the following rule: St2 is the next state after St1, if there are two stacks Stk1 and Stk2, with the block on top of Stk1 being moved to Stk2.

Search Problems: defining a transition from X to Y - example

```
% mv Top1 to Stk2 em St2
s(Stacks,[Stk1,[Top1|Stk2]|Otherstacks]) :-
% [Top1|Stk1] is a stack in St1
   del([Top1|Stk1],Stacks,Stacks1),
% Stk2 is a stack in St1
   del(Stk2,Stacks1,Otherstacks).
```

Search Problems: Hanoi

- goal (final state):

  `goal(Estado) :- member([a,b,c],Estado).`

- search predicate (can be implemented using any search predicate: dfs, bfs etc:

  `solve(Initial,Final).`

- Query: `?- solve([[c,a,b],[],[]],Solution).`

Search Problems

- Depth-first search (dfs):

```
solve(N,[N]) :- goal(N).
solve(N,[N|Sol1]) :-
    s(N,N1),        % the implementation of s/2
    solve(N1,Sol1).% depends on the problem
```

- Note: this program does not prevent cycles.

Search Problems

- Iterative Deepening: (needs extra argument: depth limit)

```
solve(N,[N],_) :- goal(N).
solve(N,[N|Sol1],ProfMax) :-
    ProfMax > 0,
    s(N,N1),
    NewMax is ProfMax - 1,
    solve(N1,Sol1,NewMax).
```

Search Problems

- Breadth-first Search (BFS):

```
bfs(Initial,Final) :- solve([[Initial]],Final).
solve([[N|Path]|_],[N|Path]) :- goal(N).
solve([Path|Paths],Solution) :-
  extend(Path,NewPaths),
  conc(Paths,NewPaths,Paths1),
  solve(Paths1,Solution).
extend([Node|Path],NewPaths) :-
  bagof([NewNode,Node|Path],
   (s(Node,NewNode), \+ member(NewNode,[Node|Path])),
    NewPaths), !.
extend(Path,_). % node has no successor.
```