State Space Search

General Problem Solving as State Space Search

In general a problem can be identified as a request for a way to get to a particular (final) situation given a current situation. In order to create a general technique for solving problems we need to formalize

the representation of the problem situation/state space

the implementation *of the available* actions/*moves that change the situation/state*

the definition *of a solution/final state*

The solution of the problem is a sequence of possible changes on the situations that lead to a final state

Representation

The representation space is a formal rendering of whatever reality the problem addresses. Usually it is simply a complex data structure that reproduces the onthology of the problem

Examples:

Puzzles:

1. The 15 puzzle : a 4X4 matrix of integers in {0,1,...,15}

2. Missionaries&Cannibals: 4 integer variable and a boolean

m-left,c-left, m-right, c-right, boat

Mathematics:

1. Finding solutions of a 2-deg equation in one variable: …

Logics:

1. Proving theorems: ...

Moves

Moves are possible actions that are available in the problem onthology. They are implemented as operations that manipulate the representation data structure.

Example:Missionaries&Cannibals

If boat=1: {m,c}-left ++; {m,c}-right --
If boat=0: {m,c}-left --; {m,c}-right ++

Final state

The final state is the desired situation to get to in order to solve the problem.

It is a condition defined on the data structure representing the problem.

Example:Missionaries&Cannibals

m-right = c-right = 3

GPS implementation ingredients

State space: struct state;

Neighborhood: SetofStates neighbors(state);

Solution: bool final (state);

Heuristic function: float H (state);

State Space Search

```
include problem
include SetofStates
path Search (state s0){
    SetOfStates horizon={s0}, explored=Æ;
     /*horizon are states that can be reached from s0;
    state view;
    while (horizon != Æ ) {
        if final((view=pick(horizon))) return(backpath(view));
        explored=+ view;
        horizon=+ (neighbors(view) - explored);
    }
    return(no solution);
}
```

State Space Search

The resulting search depends on the implementation of the funcion
state pick(set of state (horizon)

if the pick is implemented as
- a fifo we get a breadth first search
- a lifo we get a depth first search
- random we get a random search

Uninformed Search

Uninformed search strategies:

Breadth-first search BFS
Depth-first search DFS
Random search RS
Cost based search CBS
Depth-limited search DLS
Iterative deepening search IDS
Bidirectional search BiDS

Comparing Search Strategies

We want to compare search strategies relative to each other, according to the following criteria:

Completeness: does the strategy find a solution if there is one?

Time: how long does it take to find a solution?

Space: how much memory does it take to get to a solution?

Optimality: How good is the solution that the strategy finds?

Comparing Search Strategies

b is the *branching factor; d* is the *depth* of the solution found; l is the *depth limit*; m is the *maximum depth* of the search tree constructed

	BFS	CBS	DFS	DLS	IDS	BiDS
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Opt?	Yes	Yes	No	No	Yes	Yes
Comp?	Yes	Yes	No	Yes *if l>d*	Yes	Yes

Informed Search ingredients

State space: struct state;

Neighborhood: SetofStates neighbors(state);

Solution: bool final (state);

Heuristic function: float H (state);

H is a numerical function that

is an eximation of the distance (path length) of a state to a solution state

Heuristic Search

```
include problem
include SetofStates
include Heuristics
path Search (state s0){
    SetOfStates horizon={s0}, explored=Æ;
     /*horizon are states that can be reached from s0;
    state view;
    while (horizon != Æ ) {
        if final((view=minH(horizon))) return(backpath(view));
        explored=+ view;
        horizon=+ (neighbors(view) - explored);
    }
    return(no solution);
}
```