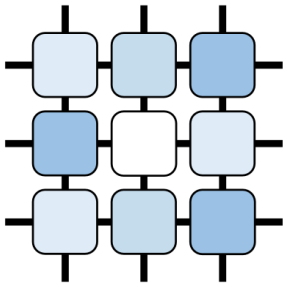
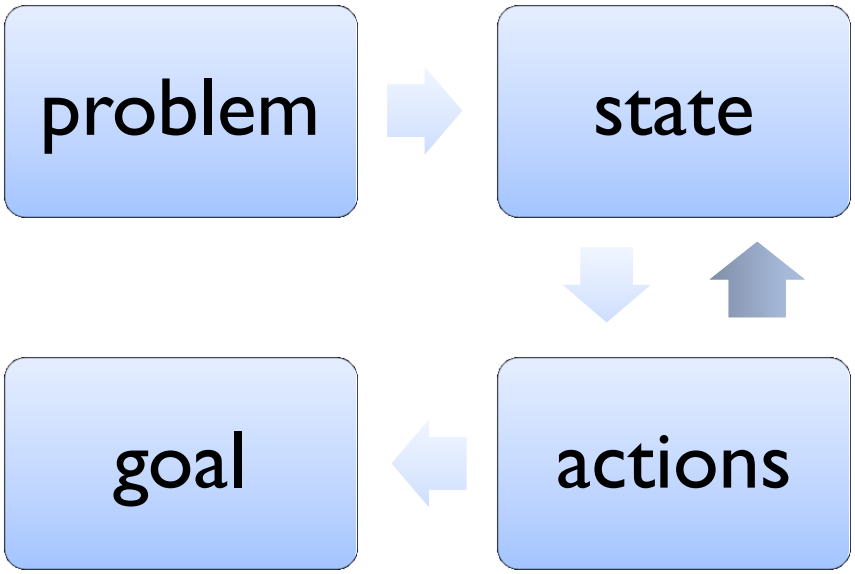


(Uninformed) State Space Search

Ondrej Vanek

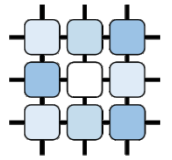
A4B33ZUI LS 2012

Problem Solving



State Space

Formulation



Problem – defined by 4 items:

Initial state – x=Karlak

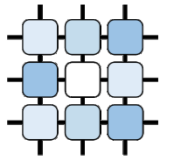
Successor function – set of action-state pairs S(x)

Goal test – explicit (x=Dejvice) vs. implicit (clean(x))

Path cost – additive (sum of distances, number of actions executed $c(x, a, y) \geq 0$)

Solution is set of actions leading from initial state to a goal state

State Space



Examples

Traveling problem

- from Karlak to Dejvice (actions – MHD)
- from Prague to Snezka (actions drive/walk)
- from Prague to Sydney (actions taxi, bus, train,plane, ferry,…)

Vacuum world

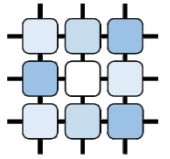
- Program your own Roomba robot!

Flight schedules for CSA

- Origin-Destination Matrix + frequency of flights
- Schedule for planes, crew – pilot + flight attendants

Tree Search Algorithm

Basic Idea



Basic idea:

offline, simulated exploration of state space

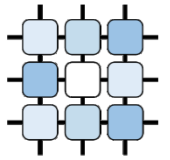
by generating successors of already-explored states

(a.k.a. expanding states)

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return the corresponding solution
    else expand the node and add the resulting nodes to the search tree
  end
```

Tree Search Algorithm

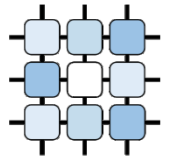
Formulation



```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE(node)) then return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

```
function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN(problem, STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

Searching the State Space



Algorithms

Breadth first search **BFS**

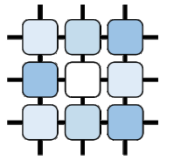
Depth first search **DFS**

Depth limited search (DFS with search limit *l*)

Iterative deepening search (Iteratively increase *l*)

Tree Search Algorithm

Formulation



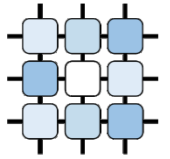
```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE(node)) then return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

BFS
Insert at the end

DFS
Insert at the beginning

```
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

Note on algorithm properties



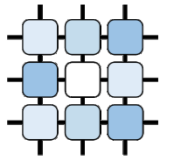
Optimal – The algorithm returns best solution.

Complete – if solution exists, the algorithm finds a solution. If not, the algorithm reports that no solution exists.

Sound – Complete and Optimal algorithm

Admissible – Optimal

Searching the State Space



Algorithms

Breadth first search **BFS**

Optimal ? Complete ?

Depth first search **DFS**

Optimal ? Complete ?

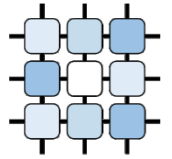
Depth limited search

Optimal ? Complete ?

Iterative deepening search

Optimal ? Complete ?

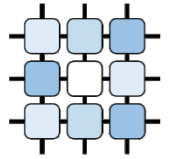
Tree Search



What problem do we have?

Graph Search

Using a closed list



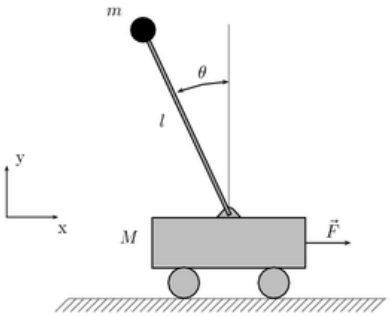
```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
    end
```

State Space

More examples

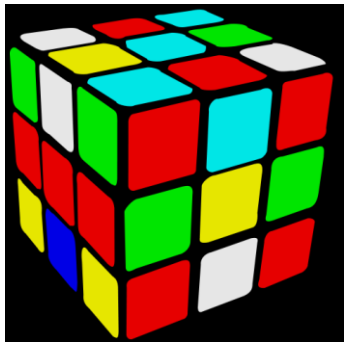
Balancing a stick (reversed pendulum)

- actions 0, L, LL, LLL, R, RR, RRR



Solving a puzzle

Rubik's cube



Monkey & Bananas

Crossword puzzles

Baking a chicken

App. Moving with friends