Searching for solutions

David Cline
Sudoku

- 9x9 grid
- Some cells filled in
- Fill in other cells so that
  - 1-9 on rows
  - 1-9 on columns
  - 1-9 on subgrids
Algorithm 1

boolea Sudoku1(Board B, int x, int y)
{
    if (y >= 9) return true;

    // Compute next X and Y location
    int xnew = (x+1)%9;
    int ynew = y + (x+1)/9;

    // This square is part of the initial placements
    if (B[y][x] > 0) {
        Sudoku1(B, xnew, ynew);
    }

    // Place every possible number
    for (val=1; val<=9; val++) {
        if (valueCanBePlaced(B, x, y, val)) {
            B[y][x] = val;
            if (Sudoku1(B, xnew, ynew)) return true;
            B[y][x] = 0;
        }
    }

    return false;
}
Algorithm 2

- Choose the square with the fewest valid numbers at each step

```java
boolean Sudoku2(Board B) {
    if (B is full) return true;

    Let (x,y) be a location with the fewest possible choices

    for (val=1; val<=9; val++) {
        if (valueCanBePlaced(B, x, y, val)) {
            B[y][x] = val;
            if (Sudoku2(B)) return true;
            B[y][x] = 0;
        }
    }
    return false;
}
```

- (Q) Discuss tradeoffs
Map of Romania
function TREE-SEARCH(problem) returns a solution, or failure  
    initialize the frontier using the initial state of problem  
    loop do  
        if the frontier is empty then return failure  
        choose a leaf node and remove it from the frontier  
        if the node contains a goal state then return the corresponding solution  
        expand the chosen node, adding the resulting nodes to the frontier  
    end loop do  

function GRAPH-SEARCH(problem) returns a solution, or failure  
    initialize the frontier using the initial state of problem  
    initialize the explored set to be empty  
    loop do  
        if the frontier is empty then return failure  
        choose a leaf node and remove it from the frontier  
        if the node contains a goal state then return the corresponding solution  
        add the node to the explored set  
        expand the chosen node, adding the resulting nodes to the frontier only if not in the frontier or explored set  
    end loop do  

Figure 3.7  An informal description of the general tree-search and graph-search algorithms. The parts of GRAPH-SEARCH marked in bold italic are the additions needed to handle repeated states.
Search Tree

(a) The initial state

(b) After expanding Arad

(c) After expanding Sibiu
Graph Search

- (Q) How do we store the explored set?
- (Q) Is this a good idea in this case?
- (Q) Suppose that the branching factor is 4 in a tree search, and 3 in a graph search. How many nodes after 10 levels for each type?
Vacuum World

- Actions are Left, Right, Suck
- (Q) What are the following:
  - states, initial state, transition model, goal test, path cost
Depth-Limited Search

function **Depth-Limited-Search**(problem, limit) returns a solution, or failure/cutoff
return **Recursive-DLS**(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function **Recursive-DLS**(node, problem, limit) returns a solution, or failure/cutoff
if problem.GOAL-Test(node.STATE) then return SOLUTION(node)
else if limit = 0 then return cutoff
else
    cutoff_occurred? ← false
    for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        result ← **Recursive-DLS**(child, problem, limit - 1)
        if result = cutoff then cutoff_occurred? ← true
        else if result ≠ failure then return result
        if cutoff_occurred? then return cutoff else return failure

**Figure 3.16** A recursive implementation of depth-limited tree search.

- (Q) Trace depth-limited search with vacuum world up to a depth of 3. Initial state: [d, dr]. Goal state [cr, c]
Iterative Deepening Search

```plaintext
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
    for depth = 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH(problem, depth)
        if result ≠ cutoff then return result
```

Figure 3.17 The iterative deepening search algorithm, which repeatedly applies depth-limited search with increasing limits. It terminates when a solution is found or if the depth-limited search returns failure, meaning that no solution exists.

• (Q) Trace iterative deepening search with vacuum:
  Initial state: [d, dr]. Goal state [cr, c]

• (Q) How many extra nodes were generated in this case?

• (Q) In a Big-O sense, how many extra nodes are generated at level k for a branching factor of n?
Breadth-First Search

function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure

  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier) /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
  Figure 3.11 Breadth-first search on a graph.

(Q) Trace breadth-first search with vacuum world up to a depth of 3. Initial state: [d, dr]. Goal state [cr, c].
Bidirectional Search

• Run 2 breadth-first searches, 1 from start, 1 from goal. Use breadth first for both sides, or iterative deepening for one side.

• Replace goal test with check to see if the frontiers of the two searches overlap.

A(Q) Trace bidirectional search for 3-position vacuum world with initial state: [d,dv,d], final state: [cv,c,c]
Math Dice

In the game Math Dice, a player attempts to create a mathematical expression from random numbers that is as close as possible to a target value. For example, given the numbers 3, 4, and 5, and a target value of 16, we can create the expression $4^{(5-3)} = 4^2 = 16$.

While it is possible to create and evaluate infix expressions directly, it is likely easier to work directly with postfix expressions. Recall that a postfix expression places the operator after the operands instead of between them. Thus, we can rewrite this expression above in postfix notation as $4 \ 5 \ 3 \ - \ ^\wedge$.

• (Q) How to represent states?
• (Q) How to generate states in the state space?
• (Q) What is the transition model?
• (Q) What is the goal test?
Postfix Generation

• (Q) Given a set of numbers and operators, how do we generate all valid postfix expressions?

• (Q) How can we generate them without duplicates?

• (Q) How can we generate them without "commutative" duplicates (3 4 +), (4 3 +)?
Postfix Evaluation

- Reading the expression from left to right
- Read a symbol s.
- If s is an operand (number), push it on the stack
- If s is an binary operator (+, -, *, /, ^)
  - pop two operands of the stack, evaluate the expression, and push the result back on the stack.
- At the end of this procedure the value of the expression will be on the top of the stack.

- (Q) Evaluate expression 3 7 + 4 *