Search

CSC 358/4578
5.27.2003
Outline

- Homework #7
- GPS
  - States and operators
  - Issues
- Search techniques
  - DFS, BFS
  - Beam search
  - A* search
  - Alpha-beta search
Homework #7

- #I
- with-list-iterator
- doiter
GPS

- General Problem Solver (GPS)
  - Developed By Alan Newell And Herbert Simon
  - 1957

- Simon
  - “It is not my aim to surprise or shock you. But the simplest way I can summarize is to say that there are now in the world machines that think, that learn, and create. Moreover, their ability to do these things is going to increase rapidly...”
GPS Cont’d

- GPS Never Lived Up To Its Expectations
- But:
  - First Program To Focus On Knowledge Representation
  - First Program To Separate Problem-Solving Strategy From Its Knowledge Of Particular Problems
GPS Cont’d

GPS Is Based On Means-Ends Analysis
- The Agent Determines What It Wants
- The Agent Determines What It Has
- The Agent Reduces The Differences Between What It Wants And What It Has
States

- The World Is Described In Terms Of States
- State Changes Are Caused By Operators That Apply To States To Produce New States
Operators

- Preconditions
  - must be true to be applicable
- Post conditions
  - Add list
  - Delete list
Problem Solving

- Problem-Solving Boils Down To Three Steps
  - Select A Domain
  - Discover A Way To Describe The Domain In Terms Of States
  - Discover A Good Set Of Operators
Implementation

- States
  - States are symbols
  - (predicate expressions)
- GPS Represents Operators As Structures:
  - Preconditions
  - Add Lists
  - Delete Lists
Basic Rules

- **Applicability**
  - The Operator Is Applicable When All Of Its Preconditions Are Satisfied.
  - A Precondition Is Satisfied When It Is Found In The Current State

- **When The Operator Is Applied**
  - All Members Of The Delete List Are Removed
  - All Members Of The Add List Are Added
Example

- Initial State:
  - (son-at-home car-needs-battery have-money have-phone-book)

- Goal State: (son-at-school)

- (GPS <initial-state> <goal-state> <operators>)
An Operator

(make-op :action 'drive-son-to-school
          :preconds '(son-at-home car-works)
          :add-list '(son-at-school)
          :del-list '(son-at-home))
GPS Algorithm

- Iterate Over A List Of Goals
  - Attempt To Achieve Them One By One
- A Goal Is Achieved If
  - It Is A Member Of The Current State
- A Goal Can Be Achieved If
  - There Is An Operator That Can Achieve It
GPS Algorithm Cont’d

(defun appropriate-p (goal op)
  (member goal (op-add-list op)))

(defun achieve (goal)
  (or (member goal *state*)
      (some #'apply-op
            (find-all goal *ops*
                :test #'appropriate-p)))
GPS Algorithm Cont’d

(defun apply-op (op)
  (when (every #'achieve (op-preconds op))
    (push (op-action op) *ops*)
    (setf *state* (set-difference *state* (op-del-list op))
          (setf *state* (union *state* (op-add-list op)))
          t))
GPS Algorithm Cont’d

(defvar *state* nil)
(defvar *ops* nil)

(defun gps (state goals ops)
  (if (every #’achieve goals)
       *ops*
       nil))
GPS Problems

- The Clobbered Sibling Goal
- Running Around The Block
- Leaping Before You Look
- Recursive Subgoal
- Sussman anomaly
- Perfect information
- Perfect action
Planning

- States are combinations of predicates
- Operators may have conditional effects
- Interleaving of planning and execution
  - replanning
Generalizing GPS: Search Tools

- Start State
- Goal State
- Successors
- Search Strategy
Tree Search

Diagram:

1

2  3
  
4 5 6 7
Tree Search, cont'd

- Main question
  - How to order the states?
(defun tree-search (states goal-p successors combiner)
  (cond ((null states) fail)
        ((funcall goal-p (first states))
          (first states))
        (t (tree-search
            (funcall combiner
              (funcall successors
                (first states))
              (rest states))
              goal-p successors combiner))))
Tree Search: Depth First Search

- Work On The Longest Paths First
- Backtrack Only When The Current State Has No More Successors

(defun depth-first-search (start goal-p successors)
  (tree-search (list start) goal-p successes #’append))
Tree Search: DFS Summary

- Depth-First Search Is OK In Finite Search Spaces
- In Infinite Search Spaces, Depth-First Search May Never Terminate
Tree Search: Breadth-First Search

- Search The Tree Layer By Layer

(defun prepend (x y) (append y x))
(defun breadth-first-search (start goal-p successors)
  (tree-search (list start) goal-p successors #'prepend))
Tree Search: BFS Summary

- In Finite Search Spaces, BFS Is Identical To DFS
- In Infinite Search Spaces, BFS Will Always Find A Solution If It Exists
- BFS Requires More Space Than DFS
Iterative Deepening

- Search depth first to level n
  - then increase n
- Seems wasteful
  - but actually is the best method for large spaces of unknown characteristics
Bi-Directional Search

- Work forwards from start
- Work backwards from goal
- Until the two points meet
Controlling Search

- Neither DFS Nor BFS Take Advantage Of Any Knowledge Of The Domain
- In Many Domains, One Can Estimate How Far The Current State Is From The Goal State
- If One Can Estimate The Distance, One Can Always Choose The Successor State That Is Closest
Best First Search

(defun sorter (cost-fn)
  #'(lambda (new old)
      (sort (append new old) #'< :key cost-fn)))

(defun best-first-search (start goal-p successors cost-fn)
  (tree-search (list start) goal-p successors
               (sorter cost-fn)))
Greedy Search

- Best = closest to goal

Problem
- Isn't guaranteed to find a solution
  - not complete
- Isn't guaranteed to find the best solution
  - not optimal
A* Search

- Best = min (path so far + estimated cost to goal)
- Restriction
  - estimate must never overestimate the cost
- If so
  - complete
  - optimal
Beam Search

- All Search Strategies So Far Keep An Ever Increasing List Of States
- DFS, BFS, And Best-First Search Are Designed To Search For A Unique Solution
- If There Are Many Solutions, One Does Not Need To Keep As Many States
- Beam Search Keeps Only A Limited Number Of States
Beam Search Cont’d

(defun beam-search (start goal-p successors cost-fn beam-width)
  (tree-search (list start) goal-p successors
               #’(lambda (old new)
                   (let ((sorted (funcall (sorter cost-fn) old new)))
                     (if (> beam-width (length sorted))
                         sorted
                         (subseq sorted 0 beam-width)))))))
Improving Beam Search

- The Width Of The Beam May Not Be Sufficient To Find A Solution
- One Can Try Different Beam Widths

(defun iter-wide-search (start goal-p successors &key (width 1) (max 100))
  (unless (> width max)
    (or (beam-search start goal-p successors cost-fn width)
         (iter-wide-search start goal-p successors cost-fn :width (+ width 1) :max max))))
Homework #8

- GPS problems