

Classical Problem Solving

EECS 344

Winter, 2008

Overview

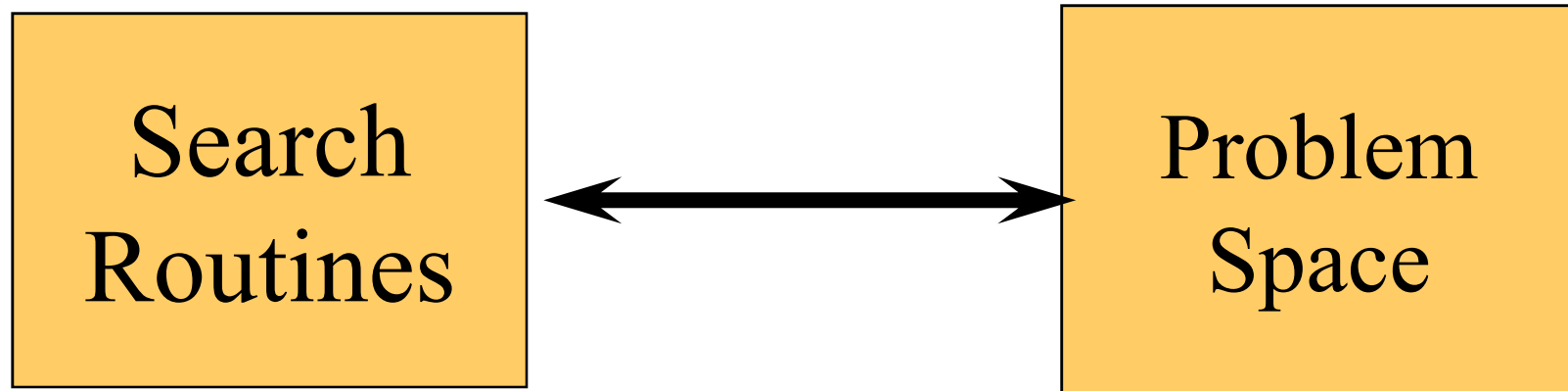
- The classical problem space model
- An implementation
 - How not to build it
 - Variations on the theme
- Examples
 - Subway navigation
 - Algebra problem solving

Early view of problem solving

- Problem solving = search in problem space
- Problem space =
 - set of *states*, representing particular situations/configurations/objects of the domain
 - set of *operators*, representing ways of generating new states from existing states.
- Problem = problem space + initial state + method for recognizing the goal

How AI would be done

- Some people define domains as problem spaces
- Other people write search routines



CPS architecture

- Define problem space/search routine interface
- Implement some search routines
- Implement some problem spaces

What operations do we need on
states?

Operations on states

- Goal detection
- Identity, to detect looping
- Display, to see the answer
- Expand it, by applying operators to it.

Implementation choices

- Define particular format for states, operators and live with it.
- Use CLOS
- Fake OOP by using procedural interface

Search Routines

- Start with breadth-first search
- Then define variations

What's wrong with this program?

```
(defun bsolve (initial)
  (bsolve1 (list (list initial initial))))
```

```
(defun bsolve1 (queue)
  (if (goal-recognizer (caar queue))
      (values (caar queue) (cdar queue))
      (bsolve1 (append (cdr queue)
                        (expand-path (car queue))))))
```

Problems

- Presumes tail-recursion
- Doesn't make paths explicit as an entity
- Doesn't provide appropriate level of debugging information
- Doesn't provide statistics

Let's look at the industrial-
strength version...

Example: Subway Navigation



Modeling subways

- Stations
 - have the lines they are on
 - x,y map coordinates
- Lines
 - have the stations on them listed

Problem Space

- state = station
- operator: Taking a line to another station
 - Can take the train to any station in one conceptual step (ignoring fares)

Let's look at the code...

Example: Solving algebraic equations

- $3X = 5X - 2$, what is X ?
- Lots of transformations one can use in solving algebra problems
- Significant novice-expert differences
- What do experts know that novices don't?

Bundy's claim

- Experts have *control knowledge* (aka metaknowledge) that lets them avoid many false paths
- Search still necessary
- But not very much

Three kinds of laws

- Attraction methods
- Collection methods
- Isolation methods

Attraction methods

- Bring occurrences of the unknown “closer together”
- Examples:
 - $WU + WV \rightarrow W(U + V)$
 - $\log(U,w) + \log(V,w) \rightarrow \log(UV,w)$

Collection methods

- Reduce the number of occurrences of the unknown
- Examples
 - $UW + UY \rightarrow U(W+Y)$
 - $(U + V)(U - V) \rightarrow U^2 - V^2$

Isolation methods

- Reduces the depth of the occurrences of the unknown
- Examples
 - $U - W = Y \rightarrow U = Y + W$
 - $\log(U, w) = Y \rightarrow U = W^Y$

Implementing Bundy's idea in CPS

- State = equation, in usual Lisp form
- Operator = procedure that
 - tests to see if the law is relevant
 - if so, proposes result of applying it.

...plus some intricate details

- An algebraic simplifier to handle “obvious” transformations

- Examples:

$$\dots + 0 + \dots \rightarrow \dots + \dots$$

$$\dots + 3 + \dots + 5 + \dots \rightarrow \dots + 8 + \dots$$

$$(U + (V + W)) \rightarrow (U + V + W)$$

Let's spelunk...

Moral of the Bundy story

- Knowledge is good. Control knowledge is especially good.
 - Careful analysis of a domain can avoid a whole lot of search
 - More leverage in knowledge than in search techniques

Assignment

- Reading: BPS, chapters 4 & 5
- Homework One: Due by start of class, 1/17/08
Problem 9, page 65. Turn in just the final system, plus the call you used to invoke it to solve part (c).