Artificial Intelligence 15-381
Heuristic Search Methods

Jaime Carbonell
jgc@cs.cmu.edu

31-August-2000

Today's Agenda

- Breadth-first search
- Bi-directional and island search
- Search Complexity
- Heuristics and evaluation functions
- Heuristic search (HC, BestFS, ...)

Complexity of Search

Definitions

Let depth d = length(min(s-Path(S_0,S_G))) - 1

Let branching-factor b = Ave(|Succ(S_i)|)

let backward b-f bb = Ave(|$Succ^{-1}$(S_i)|)

(usually b = bb, but not always)

Let C(<method>,b,d) = max number of S_i visited

C(<method>,b,d) = worst-case time complexity

C(<method>,b,d) >= worst-case space complexity

Breadth-First Search Complexity

C(BFS,b,d) = $\Sigma_{i=0,d} b^i$ = O(b^d)

C(BBFS,b,d) = $\Sigma_{i=0,d} bb^i$ = O(bb^d)

C(BiBFS,b,d) = $2\Sigma_{i=0,d/2} b^i$ = O($b^{d/2}$)
(if b = bb)

Suppose we have k evenly-spaced islands
in s-Path(S_0,S_G), then:
C(IBFS,b,d) = (k+1)$\Sigma_{i=0,d/(k+1)}$)b^i = **O**($b^{d/(k+1)}$)

C(BiIBFS,b,d) = 2(k+1)$\Sigma_{i=0,d/(2k+2)} b^i$ = **O**($b^{d/(2k+2)}$)

Heuristics in AI Search

Definition

A **Heuristic** *is an operationally-effective nugget of information on how to direct search in a problem space. Heuristics are only approximately correct.*

Common Types of Heuristics

- "If-then" rules for state-transition selection.
- Macro-operator formation [discussed later].
- Problem decomposition [e.g. hypothesizing islands on the search path].
- Estimation of distance between S_{curr} and S_G.
  (e.g. Manhattan, Eucledian, topological distance)
- Value function on each Succ(S_{curr})
- cost(path(S_0,S_{curr})) + E[cost(path(S_{curr},S_G))]
- Utility: value(S) - cost(S)

Heuristic Search

Value function: E(o-Path(S_0,S_{curr}), S_{curr}, S_G)

Since S_0 and S_G are constant, we abbreviate E(S_{curr})

General Form

1. Quit if done (with success or failure), else:
2. s-Queue := F(Succ(S_{curr}), s-Queue)
3. S_{next} := *Arg*max[E(s-Queue)]
4. Go to 1, with S_{curr} := S_{next}

Steepest ascent Hill-Climbing

- F(Succ(S_{curr}), s-Queue) = Succ(S_{curr})
- No history stored in s-Queue, hence:
  Space complexity = max(b)
  [= O(1) if b is bounded]
- Quintessential greedy search

Max-Gradient Search

- "Informed" depth-first search
- S_{next}:= *Arg*max[E(Succ(S_{curr}))]
- But if Succ(S_{next}) is null, then backtrack
- Alternative: backtrack if E(S_{next}) < E(S_{curr})

Beyond Greedy Search

Best-First Search

BestFS(S_{curr}, S_G, s-Queue)
IF $S_{curr} = S_G$, SUCCESS
For s_i in Succ(S_{curr})
   Insertion-sort(<s_i, E(s_i)>, s-Queue)
IF s-Queue = Null, FAILURE
   ELSE RETURN BestFS(FIRST(s-Queue), S_G, TAIL(s-Queue))

- F(Succ(S_{curr}), s-Queue) =
  Sort(Append(Succ(S_{curr}),Tail(s-Queue)),E(s_i))
- Full-breadth search
- "Ragged" fringe expansion
- Does BestFS guarantee optimality?

Beam Search

- ***Best-few*** BFS
- Beam-width parameter
- Uniform fringe expansion
- Does Beam Search guarantee optimality?

Macrooperators in Search

Linear Macros

Cashed sequence of instantiated operators:

If: S_0 --op_i--> S_1 --op_j--> S_2
Then: S_0 --$op_{i,j}$--> S_2

Alternative notation:
If: $op_j(op_i(S_0)) = S_2$, Then: $op_{i,j}(S_0) = S_2$

Macros can have any length, e.g. $o_{i,j,k,l,m,n}$

Key question: do linear macros reduce search?

Disjunctive Macros

```
       op_2        op_3

op_1         op_4          op_7

       op_5        op_6
```

Iterative Macros

```
                    +-----------------------------+
                    |                             |
                    V                             |
op_{i,j}---->Cond(s-Hist,S_G)---YES--->Op_{k,l,m,n}----->
                    |
                    |
                    NO
                    |
                    V
                 op_{o,p,q}
```