Classical AI

Professor Frank Kreimendahl

School of Computing and Data Science
Wentworth Institute of Technology

May 15, 2023

Search

Formalization

Representation

Basic Algorithms

Search

Formalizing Problem Solving

State: hypothetical world state
Operations: actions that modify world
Goal: desired state or test

The major features of the program that are worthy of discussion are:

1. The recursive nature of its problem-solving activity.
2. The separation of problem content from problem-solving technique as a way of increasing the generality of the program.
3. The two general problem-solving techniques that now constitute its repertoire: means-ends analysis, and planning.

Newell, Shaw, and Simon: "Report On A General Problem-Solving Program", 1959

Representation

SampleWorld: first assignment
SW state space
SW search space

Search

Basic Algorithms

Basic Algorithms

______-First Search

```
Open ← ordered list containing initial state
while true do
    if Open is empty then
        return failure
    Node ← Open.Pop()
    if Node is goal then
        return Node (or path to Node)
    else
        Children ← Expand(Node)
        Add Children to front of Open
```

DFS Evaluation

Assume branching factor b and solution depth d

Completeness:
Time:
Space:
Admissibility:

Breadth-First Search

```
Open ← ordered list containing initial state
while true do
    if Open is empty then
        return failure
    Node ← Open.Pop()
    if Node is goal then
        return Node (or path to Node)
    else
        Children ← Expand(Node)
        Add Children to end of Open
```

BFS Evaluation

Assume branching factor b and solution depth d

Completeness:

Time:

Space:

Admissibility:

Uniform-Cost Search

```
Open ← priority queue containing initial state
while true do
    if Open is empty then
        return failure
    Node ← Open.Pop()
    if Node is goal then
        return Node (or path to Node)
    else
        Children ← Expand(Node)
        Add Children to Open, sorted by path cost
```

- Check for cycles with *Node*'s ancestors
- Maintain **closed** list to detect duplicates

Comparisons

Algorithm	Time	Space	Complete	Admissible
DFS	b^m	bm	if $m \geq d$	no
BFS	b^d	b^d	yes	if ops cost 1
Uniform-cost	b^d	b^d	yes	yes

b: branching factor
d: solution depth
m: maximum explored depth

Resource Requirements for Solutions

Assume $b = 10$, 100k nodes/sec, 100 bytes/node

Sol. depth	Nodes	Time	Space
1	11	.11 ms	1.1Kb
2	111	1.1 ms	11Kb
4	11,111	.11 sec	1Mb
6	10^6	11 sec	111Mb
8	10^8	18 min	11Gb
10	10^{10}	31 hours	1Tb
12	10^{12}	128 days	111Tb
14	10^{14}	35 years	11Pb

Search Tradeoffs

BFS is complete and admissible, but uses b^d space

DFS uses bd space, but is not admissible and only complete if $m > d$

How can we get the best of both algorithms?

Iterative Deepening Search

for $d = 1$ to ∞ **do**
    DFS to level d
    if DFS succeeds **then**
        return solution

IDS Evaluation

Assume branching factor b and solution depth d

Completeness:
Time:
Space:
Admissibility:

Runtime

How can this be efficient? This generates nodes near the start many times.

Intuition: Because of branching, most of the generated nodes are at the same depth as the goal node.

generated nodes $= b^d + 2b^{d-1} + \cdots + (d-1)b^2 + db$