Lecture 7
Search Wrap Up,
Intro to Constraint Satisfaction Problems
Lecture Overview

- Recap of previous lecture
  - Dynamic programming for search
  - Other advanced search algorithms
  - Intro to CSP (time permitting)
A* properties

We showed that A* is optimal and complete, under certain conditions
A* properties

We showed that A* is optimal and complete, under certain conditions

Which of the following conditions is not needed?

A. Arc costs are bounded above 0

B. Branching factor is finite

C. \( h(n) \) is an underestimate of the cost of the shortest path from \( n \) to a goal

D. The costs around a cycle must sum to zero
We showed that A* is optimal and complete, under certain conditions.

Which of the following conditions is not needed?

A. Arc costs are bounded above 0

B. Branching factor is finite

C. $h(n)$ is an underestimate of the cost of the shortest path from $n$ to a goal

D. The costs around a cycle must sum to zero
Cycle Checking

- If we want to get rid of cycles, but we also want to be able to find multiple solutions
  - Do **cycle checking**

- In DFS-type search algorithms
  - We can do cheap cycle checks: as low as constant time (i.e. independent of path length)

- In BFS-type search algorithms
  - Cycle checking requires time linear in the length of the expanded path
If we only want one path to the solution

- Can prune path to a node \( n \) that has already been reached via a previous path
  - Subsumes cycle check
- Must make sure that we are not pruning a shorter path to the node
  - Don’t need to do this with LCSF:
    - always finds the shortest path to any node of the frontier first
  - We saw that A* is not guaranteed to do so
    - However, there are special conditions on the \( h(n) \) that can recover the guarantee of LCFS for A*: the *monotone restriction* (See P&M text, Section 3.7.2, not required for this course)
Branch-and-Bound Search

One way to combine DFS with heuristic guidance

• Follows exactly the same search path as depth-first search
  • But to ensure optimality, it does not stop at the first solution found

• It continues, after recording upper bound on solution cost
  • upper bound: $UB = \text{cost of the best solution found so far}$
  • Initialized to $\infty$ or any overestimate of optimal solution cost

• When a path $\rho$ is selected for expansion:
  • Compute lower bound $LB(\rho) = f(\rho)$
    - If $LB(\rho) \geq UB$, remove $\rho$ from frontier without expanding it (pruning)
    - Else expand $\rho$, adding all of its neighbors to the frontier
A* properties

We showed that B&B is optimal and complete, under certain conditions

Which of the following conditions is not needed?

A. The upper bound must be initialized to a finite overestimate of the solution cost

C. $h(n)$ is an underestimate of the cost of the shortest path from $n$ to a goal

D. The lower bound set based on path $p$, $LB(p)$, must always be an underestimate of the actual cost of getting to the goal through $p$.

E. All of the above are needed
A* properties

We showed that B&B is optimal and complete, under certain conditions

Which of the following conditions is not needed?

A. The upper bound must be initialized to an finite overestimate of the solution cost

C. \( h(n) \) is an underestimate of the cost of the shortest path from \( n \) to a goal

EITHER OF THESE, they are equivalent so only one of the two is needed

D. The upper bound set based on path \( p \) must always be an underestimate of the actual cost of getting to the goal through \( p \).

E. All of the above are needed
## Search Methods so Far

<table>
<thead>
<tr>
<th></th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DFS</strong></td>
<td>N</td>
<td>N</td>
<td>(O(b^m))</td>
<td>(O(mb))</td>
</tr>
<tr>
<td><strong>BFS</strong></td>
<td>Y</td>
<td>Y</td>
<td>(O(b^m))</td>
<td>(O(b^m))</td>
</tr>
<tr>
<td><strong>IDS</strong></td>
<td>Y</td>
<td>Y</td>
<td>(O(b^m))</td>
<td>(O(mb))</td>
</tr>
<tr>
<td><strong>LCFS</strong> (when arc costs available)</td>
<td>Y (Costs &gt; 0)</td>
<td>Y (Costs &gt;=0)</td>
<td>(O(b^m))</td>
<td>(O(b^m))</td>
</tr>
<tr>
<td><strong>Best First</strong> (when (h) available)</td>
<td>N</td>
<td>N</td>
<td>(O(b^m))</td>
<td>(O(b^m))</td>
</tr>
<tr>
<td><strong>A</strong>^* (when arc costs &gt; 0 and (h) admissible)</td>
<td>Y</td>
<td>Y</td>
<td>(O(b^m))</td>
<td><strong>Optimally Efficient</strong></td>
</tr>
<tr>
<td><strong>Branch-and-Bound</strong> (Y with finite initial bound)</td>
<td>N</td>
<td>Y (If (h) admissible)</td>
<td>(O(b^m))</td>
<td>(O(bm))</td>
</tr>
</tbody>
</table>
Lecture Overview

• Recap of Lecture 9

Dynamic programming for search
• Other advanced search algorithms

• Intro to CSP (time permitting)
Dynamic Programming

- Idea: for statically stored graphs, build a table of dist(n):
  - The actual distance of the shortest path from node n to a goal g
  - This is the perfect ...........

- dist(g) =
- dist(z) =
- dist(c) =
- dist(b) =
Dynamic Programming

- Idea: for statically stored graphs, build a table of dist(n):
  - The actual distance of the shortest path from node n to a goal g
  - This is the perfect ...........

- dist(g) = 0
- dist(z) = 1
- dist(c) = 3
- dist(b) = 4
- dist(k) = ?

A. 6  B. 7  C. ∞
Dynamic Programming

- Idea: for statically stored graphs, build a table of dist(n):
  - The **actual distance** of the **shortest path** from node n to a goal g
  - This is the perfect h(h)

- dist(g) = 0
- dist(z) = 1
- dist(c) = 3
- dist(b) = 4
- dist(k) = 6

- dist(m) = ?
  - A. 6
  - B. 7
  - C. ∞
Dynamic Programming

• Idea: for statically stored graphs, build a table of dist(n):
  • The actual distance of the shortest path from node n to a goal g
  • This is the perfect h(h)

• dist(g) = 0
• dist(z) = 1
• dist(c) = 3
• dist(b) = 4
• dist(k) = 6
• dist(m) = ?

A. 6  B. 7  C. ∞
Dynamic Programming

Dist(n) can be built backwards from the goal:

\[ dist(n) = \begin{cases} 
0 & \text{if is \_ goal(n),} \\
\min_{(n,m) \in A} (\text{cost}(n,m) + dist(m)) & \text{otherwise}
\end{cases} \]

over all the neighbors m of n

Slide 17
Dynamic Programming

Dist(n) can be built **backwards** from the goal:

\[
dist(n) = \begin{cases} 
  0 & \text{if \ is\_goal(n),} \\
  \min_{\langle n,m \rangle \in A} (\text{cost}(n,m) + dist(m)) & \text{otherwise} 
\end{cases}
\]

over all the neighbors m of n

Dist(KD) =
Dynamic Programming

Dist(n) can be built **backwards** from the goal:

\[
\text{dist}(n) = \begin{cases} 
0 & \text{if } \text{is\_goal}(n), \\
\min_{(n,m) \in A} \left( \text{cost}(n,m) + \text{dist}(m) \right) & \text{otherwise} 
\end{cases}
\]

over all the neighbors m of n

Dist(KD) = min [(4+8), (3+8)] = 11

Requires **explicit goal nodes** (the previous methods only assumed a function that recognizes goal nodes);
Dynamic Programming

• Idea: for *statically stored graphs*, build a table of $\text{dist}(n)$:
  • The *actual distance* of the *shortest path* from node $n$ to a goal $g$
  • This is the perfect $h$

• How could we implement that?
  • Run one of the search algorithms we have seen so far in the *backwards* graph (arcs reversed), starting from the goal
  • Can create the backwards graph in Aispace by using *invert graph*, in *create* mode
Dynamic Programming

- Idea: for **statically stored graphs**, build a table of \( \text{dist}(n) \):
  - The **actual distance** of the **shortest path** from node \( n \) to a goal \( g \)
  - This is the perfect \( h \)

- How could we implement that?
  - Run one of the search algorithms we have seen so far in the **backwards** graph (arcs reversed), starting from the goal
  - Can create the backwards graph in Aispace by using \textit{invert graph}, in \textit{create} mode

Which algorithm should we use?

A. \( A^* \)

B. LCSF

C. LCSF with multiple path pruning

D. Best First Search
Dynamic Programming

- Idea: for **statically stored graphs**, build a table of dist(n):
  - The **actual distance** of the **shortest path** from node n to a goal g
  - This is the perfect h

- How could we implement that?
  - Run LCFS with multiple path pruning in the **backwards** graph (arcs reversed), starting from the goal
  - Can create the backwards graph in Aispace by using *invert graph*, in *create* mode

Which algorithm should we use?

C. LCSF with multiple path pruning

It is the only one guaranteed to find the shortest path to the goal without needing h(n)

We want multiple path pruning because we are only interested in the first, shortest path from each node to s
Dynamic Programming

• Idea: for statically stored graphs, build a table of dist(n):
  • The actual distance of the shortest path from node n to a goal g
  • This is the perfect h

• How could we implement that?
  • Run LCFS with multiple path pruning in the backwards graph (arcs reversed), starting from the goal
  • Can generate the backward graph in AISpace by using invert graph, in create mode
LCSF on Inverted Graph (sample steps)
Dynamic Programming

- Idea: for statically stored graphs, build a table of dist(n):
  - The **actual distance** of the shortest path from node n to a goal g
  - This is the perfect h

- How could we implement that?
  - Run LCFS with multiple path pruning in the **backwards** graph (arcs reversed), starting from the goal
  - Can do this in Aispace by using *invert graph*, in *create* mode

- When it’s time to act (forward): for each node n always pick neighbor m that minimizes
  \[
  \min_{(n,m) \in A} (\text{cost}(n,m) + \text{dist}(m))
  \]

- Problems?
Dynamic Programming

• Idea: for statically stored graphs, build a table of dist(n):
  • The actual distance of the shortest path from node n to a goal g
  • This is the perfect h

• How could we implement that?
  • Run LCFS with multiple path pruning in the backwards graph (arcs reversed), starting from the goal
  • Can do this in Aispace by using invert graph, in create mode

• When it’s time to act (forward): for each node n always pick neighbor m that minimizes
  \[
  \min_{(n,m) \in A} (\text{cost}(n,m) + \text{dist}(m))
  \]

• Problems?
  • Needs space to explicitly store the full search graph
  • The \text{dist} function needs to be recomputed for each goal: method not suitable when goals vary often
Lecture Overview

• Recap of Lecture 9
• Dynamic programming for search
• Other advanced search algorithms
• Intro to CSP (time permitting)
Iterative Deepening A* (IDA*)

Branch & Bound (B&B) can still get stuck in infinite (or extremely long) paths

• Search depth-first, but to a fixed depth, as we did for Iterative Deepening
Iterative Deepening DFS (IDS) in a Nutshell

- Use DFS to look for solutions at depth 1, then 2, then 3, etc
  - For depth D, ignore any paths with longer length
  - Depth-bounded depth-first search

If no goal re-start from scratch and get to depth 2

If no goal re-start from scratch and get to depth 3

If no goal re-start from scratch and get to depth 4
Which of the following is **false**:  

A. Iterative deepening saves space over breadth-first search  
B. Iterative deepening finds same answer as breadth-first search  
C. Iterative deepening runs faster than breadth-first search  
D. Iterative deepening recomputes elements of the frontier that breadth-first search stores
Which of the following is false:

A. Iterative deepening saves space over breadth-first search
B. Iterative deepening finds same answer as breadth-first search
C. Iterative deepening runs faster than breadth-first search
D. Iterative deepening recomputes elements of the frontier that breadth-first search stores
Iterative Deepening A* (IDA*)

- Like Iterative Deepening DFS
  - But the “depth” bound is measured in terms of $f$
  - IDA* is a bit of a misnomer
    - The only thing it has in common with A* is that it uses the f value
      $$f(p) = \text{cost}(p) + h(p)$$
    - It does NOT expand the path with lowest f value. It is doing DFS!
    - But f-value-bounded DFS doesn’t sound as good ...
- Start with f-value = $f(s)$ (s is start node)
- If you don’t find a solution at a given f-value
  - Increase the bound: to the minimum of the f-values that exceeded the previous bound
- Will explore all nodes n with f value $\leq f \text{ min}$ (optimal one)
  - Under the same conditions for the optimality of A*
Numbers inside nodes are their f scores

The algorithm would have started with a bound of 1 (f of the start state).

The current bound of 3 is the minimum of the f values found to exceed bound = 1 (i.e. 3 and 4) in that iteration.
f values found to exceed the bound of 4 in this iteration

Paths are expanded up to this threshold

Threshold = 4
Threshold = 5

6, 7, 12

Paths are expanded up to this threshold
Threshold = 6

Paths are expanded up to this threshold

11,13,7,12
Analysis of Iterative Deepening A* (IDA*)

- Complete and optimal?
- Time complexity:
- Space complexity:
Analysis of Iterative Deepening A* (IDA*)

- Complete and optimal? Yes, under the same conditions as A*
- Time complexity: $O(b^m)$
  - Same as DFS, even though we visit paths multiple times (see slides on uninformed IDS)
- Space complexity: $O(bm)$
  - Same as DFS and IDS
- Compared to Branch and Bound:
  - Advantage:
  - Disadvantage:
Analysis of Iterative Deepening A* (IDA*)

- Complete and optimal? Yes, under the same conditions as A*
- Time complexity: $O(b^m)$
  - Same as DFS, even though we visit paths multiple times (see slides on uninformed IDS)
- Space complexity: $O(bm)$
  - Same as DFS and IDS
- Compared to Branch and Bound:
  - Advantages
    - does not need a finite overestimate of the solution cost to be complete
    - Does not need to keep searching after finding a solution
  - Disadvantage: multiple re-expansions of nodes
## Search Methods so Far

<table>
<thead>
<tr>
<th>Method</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DFS</strong></td>
<td>N</td>
<td>N</td>
<td>$O(b^m)$</td>
<td>$O(mb)$</td>
</tr>
<tr>
<td><strong>BFS</strong></td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td><strong>IDS</strong></td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>$O(mb)$</td>
</tr>
<tr>
<td><strong>LCFS</strong> (when arc costs available)</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td></td>
<td>Costs &gt; 0</td>
<td>Costs &gt;=0</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Best First</strong> (when $h$ available)</td>
<td>N</td>
<td>N</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td><strong>A</strong> (when arc costs &gt; 3 and $h$ admissible)</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Optimally Efficient</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Branch-and-Bound</strong> (Y with finite initial bound)</td>
<td>N</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>$O(bm)$</td>
</tr>
<tr>
<td><strong>IDA</strong></td>
<td></td>
<td></td>
<td></td>
<td>40</td>
</tr>
</tbody>
</table>
# Search Methods so Far

<table>
<thead>
<tr>
<th>Method</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DFS</strong></td>
<td>N</td>
<td>N</td>
<td>$O(b^m)$</td>
<td>$O(mb)$</td>
</tr>
<tr>
<td><strong>BFS</strong></td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td><strong>IDS</strong></td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>$O(mb)$</td>
</tr>
<tr>
<td><strong>LCFS</strong> (when arc costs available)</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td><strong>Best First</strong> (when $h$ available)</td>
<td>N</td>
<td>N</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td><strong>A</strong> (when arc costs &gt; 3 and $h$ admissible)</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>Optimally Efficient $O(b^m)$</td>
</tr>
<tr>
<td><strong>Branch-and-Bound</strong> (Y with finite initial bound)</td>
<td>N</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>$O(bm)$</td>
</tr>
<tr>
<td><strong>IDA</strong> (when arc costs &gt; 0 and $h$ admissible)</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>$O(bm)$</td>
</tr>
</tbody>
</table>

- **uninformed**
- **Informed (goal directed)**
- **Uninformed but using arc cost**
Heuristic DFS

• Other than IDA*, how else can we use heuristic information in DFS?
Heuristic DFS

• Other than IDA*, how else we use heuristic information in DFS?
  • When we expand a node, we put all its neighbours on the frontier
  • In which order? Matters because DFS uses a LIFO stack
    ✔ Can use heuristic guidance: h or f
    ✔ Perfect heuristic f: would solve problem without any backtracking

• Heuristic DFS is very frequently used in practice
  • Simply choose promising branches first
  • Based on any kind of information available
Heuristic DFS

• Other than IDA*, how else we use heuristic information in DFS?
  • When we expand a node, we put all its neighbours on the frontier
  • In which order? Matters because DFS uses a LIFO stack
    ✓ Can use heuristic guidance: h or f
    ✓ Perfect heuristic f: would solve problem without any backtracking

• Heuristic DFS is very frequently used in practice
  • Simply choose promising branches first
  • Based on any kind of information available
    Does it have to be admissible?

A. Yes  B. No  C. It depends
Heuristic DFS

• Other than IDA*, how else we use heuristic information in DFS?

  • When we expand a node, we put all its neighbours on the frontier

  • In which order? Matters because DFS uses a LIFO stack

    ✓ Can use heuristic guidance: h or f

    ✓ Perfect heuristic f: would solve problem without any backtracking

• Heuristic DFS is very frequently used in practice

  • Simply choose promising branches first

B. No
We are still doing DFS, i.e. following each path all the way to end before trying any other.
Heuristic DFS and More

• Can we combine heuristic DFS with IDA*?
Heuristic DFS and More

• Can we combine this with IDA*?
  • DFS with an f-value bound (using admissible heuristic h)
  • putting neighbors onto frontier in a smart order (using some heuristic h’)
  • Can, of course, also choose h’ = h
Memory-bounded $A^*$

- Iterative deepening $A^*$ and B & B use little memory
- What if we have some more memory (but not enough for regular $A^*$)?
  - Do $A^*$ and keep as much of the frontier in memory as possible
  - When running out of memory
    - delete worst paths (highest $f$) from frontier (e.g. $p_1$, ..$p_n$ below)
    - Backup the value of the deleted paths to a common ancestor (e.g. $N$ below)
      - Way to remember the potential value of the “forgotten” paths

The corresponding subtrees get regenerated only when all other paths have been shown to be worse than the “forgotten” path
MBA*: Compute New $h(p)$

If we want to prune subpaths $p_1$, $p_2$, .., $p_n$ below and “back up” their value to common ancestor $p$

\[
\text{New } h(p) = \max \left( \min \left[ (\text{cost}(p_i) - \text{cost}(p)) + h(p_i) \right], \quad \text{Old } h(p) \right)
\]

$(\text{cost}(p_i) - \text{cost}(p)) + h(p_i)$ gives the estimated cost of the pruned subpath from $p$ to $p_i$

$\min \left[ (\text{cost}(p_i) - \text{cost}(p)) + h(p_i) \right]$ gives the pruned subpath with the most promising estimated cost

Taking the max with $\text{Old } h(p)$ gives the tighter $h$ value for $p$
Memory-bounded $A^*$

Details of the algorithm are beyond the scope of this course but

- It is **complete**, if there is any **reachable solution**, i.e. a solution at a depth manageable by the available memory
- It is **optimal** if the optimal solution is reachable
  - Otherwise it returns the best reachable solution given the available memory
- Often used in practice: considered one of the best algorithms for finding optimal solutions under memory limitations
- It can be bogged down by having to switch back and forth among a set of candidate solution paths, of which only a few fit in memory
# Recap (Must Know How to Fill This)

<table>
<thead>
<tr>
<th></th>
<th>Selection</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BFS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCFS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Best First</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B&amp;B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDA*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MBA*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithm</td>
<td>Selection</td>
<td>Complete</td>
<td>Optimal</td>
<td>Time</td>
<td>Space</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>----------</td>
<td>---------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>DFS</td>
<td>LIFO</td>
<td>N</td>
<td>N</td>
<td>$O(b^m)$</td>
<td>$O(mb)$</td>
</tr>
<tr>
<td>BFS</td>
<td>FIFO</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>IDS</td>
<td>LIFO</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>$O(mb)$</td>
</tr>
<tr>
<td>LCFS</td>
<td>min cost</td>
<td>Y **</td>
<td>Y **</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>Best First</td>
<td>min h</td>
<td>N</td>
<td>N</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>A*</td>
<td>min f</td>
<td>Y**</td>
<td>Y**</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>B&amp;B</td>
<td>LIFO + pruning</td>
<td>Y**</td>
<td>Y**</td>
<td>$O(b^m)$</td>
<td>$O(mb)$</td>
</tr>
<tr>
<td>IDA*</td>
<td>LIFO</td>
<td>Y**</td>
<td>Y**</td>
<td>$O(b^m)$</td>
<td>$O(mb)$</td>
</tr>
<tr>
<td>MBA*</td>
<td>min f</td>
<td>Y**</td>
<td>Y**</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
</tbody>
</table>

** Needs conditions: you need to know what they are.
### Algorithms Often Used in Practice

<table>
<thead>
<tr>
<th>Selection</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>LIFO</td>
<td>N</td>
<td>N</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>BFS</td>
<td>FIFO</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>IDS</td>
<td>LIFO</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>LCFS</td>
<td>min cost</td>
<td>Y **</td>
<td>Y **</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>Best First</td>
<td>min h</td>
<td>N</td>
<td>N</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>A*</td>
<td>min f</td>
<td>Y**</td>
<td>Y**</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>B&amp;B</td>
<td>LIFO + pruning</td>
<td>Y**</td>
<td>Y**</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>IDA*</td>
<td>LIFO</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>MBA*</td>
<td>min f</td>
<td>Y**</td>
<td>Y**</td>
<td>$O(b^m)$</td>
</tr>
</tbody>
</table>

** Needs conditions: you need to know what they are
Search in Practice

Many paths to solution, no ∞ paths?

Informed? NO

B&B

IDS

Large branching factor? NO

Y

IDA*

MBA*

These are indeed general guidelines, specific problems might yield different choices
Remember Deep Blue?

Deep Blue’s Results in the second tournament:

- second tournament: won 3 games, lost 2, tied 1

- 30 CPUs + 480 chess processors
- Searched 126.000.000 nodes per sec
- Generated 30 billion positions per move reaching depth 14 routinely

- Iterative Deepening with evaluation function (similar to a heuristic) based on 8000 features (e.g., sum of worth of pieces: pawn 1, rook 5, queen 10)
Sample applications

• An Efficient A* Search Algorithm For Statistical Machine Translation. 2001 (DMMT '01 Proceedings of the workshop on Data-driven methods in machine translation - Volume 14 )

  • Machine Vision ... Here we consider a new compositional model for finding salient curves.

• Factored A*search for models over sequences and trees. IJCAI 2003
  • It starts by saying... The primary challenge when using A* search is to find heuristic functions that simultaneously are admissible, close to actual completion costs, and efficient to calculate...
  • applied to NLP and BioInformatics

• Recursive Best-First Search with Bounded Overhead (AAAI 2015)
  • “We show empirically that this improves performance in several domains, both for optimal and suboptimal search, and also yields a better linear-space anytime heuristic search. RBFSCR is the first linear space best-first search robust enough to solve a variety of domains with varying operator costs.”
Learning Goals for search

- **Identify** real world examples that make use of deterministic, goal-driven search agents
- **Assess** the size of the search space of a given search problem.
- **Implement** the generic solution to a search problem.
- **Apply** basic properties of search algorithms:
  - completeness, optimality, time and space complexity of search algorithms.
- **Select** the most appropriate search algorithms for specific problems.
- **Define/read/write/trace/debug** different the search algorithms covered
- **Implement** cycle checking and multiple path pruning for different algorithms
  - Identify when they are appropriate
- **Construct** heuristic functions for specific search problems
- **Formally prove** A* optimality.
- **Understand** general ideas behind Dynamic Programming and MBA*