Search

Uninformed Algorithms

Introduction to Artificial Intelligence

Chandra Gummaluru
University of Toronto

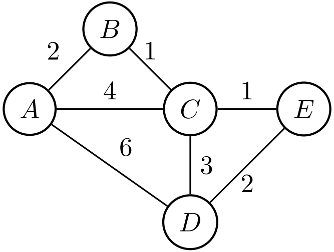
Version W22.1

- The following is based on material developed by many individuals, including (but not limited to):
  - Sheila McIlraith
  - Bahar Aameri
  - Fahiem Bacchus
  - Sonya Allin

- In the previous module, we claimed that the order in which the successors are explored (i.e., the removal order) changes the search algorithm's properties.

- In this module, we will consider three orderings:
  1. first-in-first-out (breadth-first search / BFS)
  2. first-in-last-out (depth-first search / DFS)
  3. smallest cumulative cost first (uniform cost search / UCS)

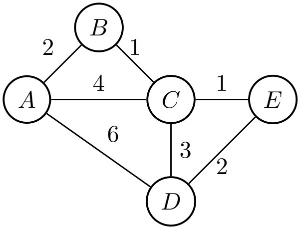
- To compare the effects of different orderings, we will run the corresponding algorithms on the following search graph (starting at A and looking for E):



- We will use local path-checking and break ties alphabetically.

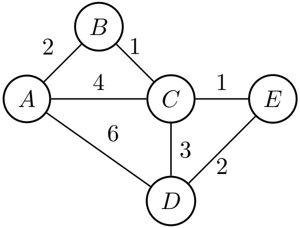
- In breadth-first search (BFS), the open is treated as a queue. In other words, paths discovered less recently are explored first.

Itr.	p	$\mathcal{O}$
1	–	A
2	A	AB, AC, AD
3	AB	AC, AD, ABC
4	AC	AD, ABC, ACB, ACD, ACE
5	AD	$ABC, ACB, ACD, ACE, ADC, ADE$
6	ABC	$ACB, ACD, ACE, ADC, ADE, ABCD, ABCE$
7	ACB	$ACD, ACE, ADC, ADE, ABCD, ABCE$
8	ACD	$ACE, ADC, ADE, ABCD, ABCE, ACDE$
9	ACE	$ADC, ADE, ABCD, ABCE, ACDE$



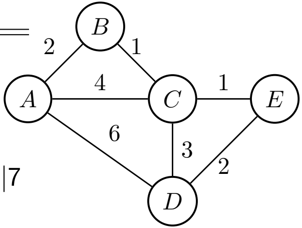
- In depth-first search (DFS), the open is treated as a stack. In other words, paths discovered more recently are explored first.

Itr.	p	$\mathcal{O}$
1	–	A
2	A	AB, AC, AD
3	AB	ABC, AC, AD
4	ABC	$ABCD, ABCE, AC, AD$
5	$ABCD$	$ABCDE, ABCE, AC, AD$
6	$ABCDE$	$ABCE, AC, AD$



- In uniform-cost search (UCS), the open is treated as a heap, ordered by path costs. In other words, the cheapest paths are explored first.

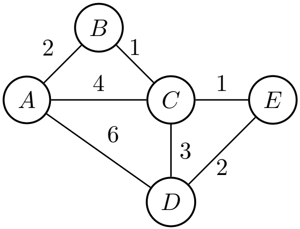
Itr.	p	$\mathcal{O}$
1	–	$A\|0$
2	A	$AB\|2, AC\|4, AD\|6$
3	AB	$ABC\|3, AC\|4, AD\|6$
4	ABC	$AC\|4, ABCE\|4, ABCD\|6, AD\|6$
5	AC	$ABCE\|4, ACB\|5, ACE\|5, ABCD\|6, AD\|6, ACD\|7$
6	$ABCE$	$ACB\|5, ACE\|5, ABCD\|6, AD\|6, ACD\|7$



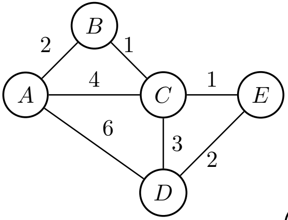
Uniform-Cost Search Search: Example (with Global Path Checking)

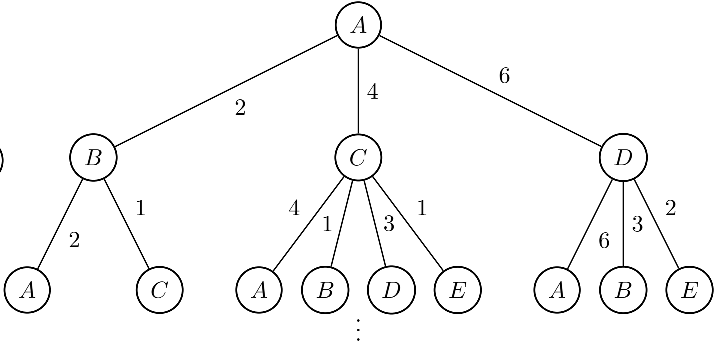
- It turns out that UCS satisfies the condition needed to implement global path checking (we will prove this later).

Itr.	p	$\mathcal{O}$
1	–	$A\|0$
2	A	$AB\|2, AC\|4, AD\|6$
3	AB	$ABC\|3, AC\|4, AD\|6$
4	ABC	$AC\|4, ABCE\|4, ABCD\|6, AD\|6$
5	AC	$ABCE\|4, ACE\|5, ABCD\|6, AD\|6, ACD\|7$
6	$ABCE$	$ACE\|5, ABCD\|6, AD\|6, ACD\|7$



- Recall that the generalized search algorithm explores a tree of possible paths.





- We wish to determine the properties of the various search algorithm in terms of b, d, ε, m and c^*, where:
  - b, m, and ε, be the branching factor, depth, and minimum edge weight of the tree
  - d and c^* denote the length and cost of the optimal solution

- **Complete** for $b < \infty$.
  - There are at most b^l paths of length, l.
  - All paths of length l are explored before any path of length $l + 1$.
  - If the solution path, p, is the last path of length d to be explored, the maximum number of nodes generated before p is

    $$1 + b + b^2 + \cdots + b^d + b(b^d - 1),$$

    which is finite for $d < \infty$.
- **Optimal** if $c(\cdot)$ is constant.
  - Action costs are not used by the algorithm.
- **Time Complexity**: $O(b^{d+1})$.
  - The maximum number of nodes generated is $1 + b + b^2 + \cdots + b^d + b(b^d - 1)$.
- **Space Complexity**: $O(b^{d+1})$.
  - The maximum number of nodes simultaneously on the open is $b(b^d - 1)$.

- **Complete** for $b, m < \infty$.
  - If b and m are both finite, then the tree of possible paths contains a finite number of paths, namely, $1 + b + b^2 + \ldots b^m$.
- **Suboptimal**.
  - Action costs are not used by the algorithm.
- **Time Complexity**: $O(b^m)$.
  - In the worst case, the search will need to explore all $1 + b + b^2 + \cdots + b^m$ paths.
- **Space Complexity**: $O(bm)$.
  - Whenever a node is expanded, at most b other nodes, $t_1, \ldots, t_b$, will be discovered.
  - All successors of t_i must be explored before any successor of t_j is discovered.
  - The maximum number of times a node can be expanded is m.
  - Thus, the maximum number of nodes simultaneously on the open is at most bm.

- **Complete** for $b < \infty$.
  - We will prove this later.
- **Optimal** always.
  - We will prove this later.
- **Time Complexity**: $O(b^{c^*/\varepsilon+1})$.
  - We use the result from the BFS analysis and the fact that the maximum length of the solution, d, is c^*/ε.
- **Space Complexity**: $O(b^{c^*/\varepsilon+1})$.
  - We use the result from the BFS analysis and the fact that the maximum length of the solution, d, is c^*/ε.

- Below, we summarize the properties of BFS, DFS, and UCS:

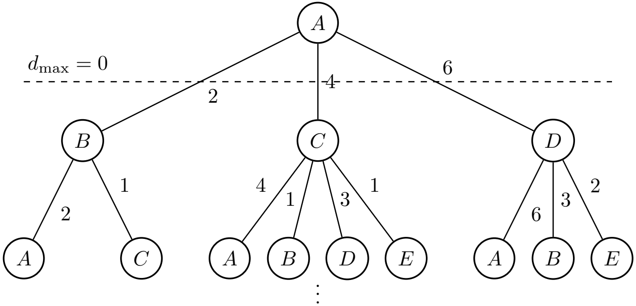
Property	BFS	DFS	UCS
Complete	$b < \infty$	$b, m < \infty$	$b < \infty$
Optimal	constant c	never	always
Time Complexity	$O(b^{d+1})$	$O(b^m)$	$O(b^{c^*/\varepsilon+1})$
Space Complexity	$O(b^{d+1})$	$O(bm)$	$O(b^{c^*/\varepsilon+1})$

- Note that $d \leq m$, and typically, $d \ll m$. Thus, generally speaking, BFS is time-efficient while DFS is space-efficient.

- Unlike with BFS, the completeness of DFS requires that m be finite. Otherwise, DFS can get stuck exploring an infinitely long path.
- If we knew d ahead of time, we could enforce a depth limit so that DFS only expands paths whose length is strictly less than d.
- Otherwise, we could start with a depth limit of $d_{\text{max}} = 0$, and repeatedly perform DFS, increasing the depth limit each time, until a solution is found.
- This is called **iterative-deepening depth-first search** (IDDFS).

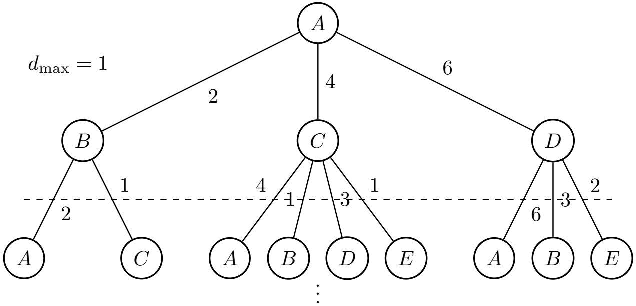
- IDDFS is like performing DFS multiple times on a sub-tree of possible paths.

Itr.	p	$\mathcal{O}$
1	–	A
2	A	



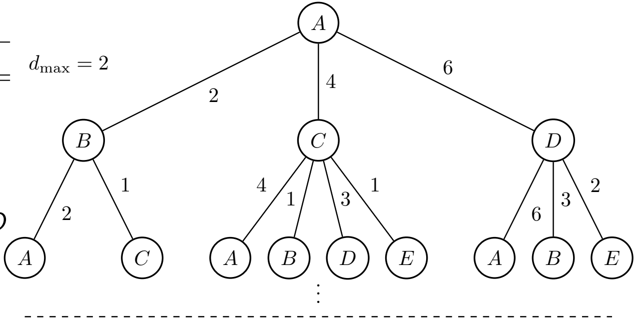
- IDDFS is like performing DFS multiple times on a sub-tree of possible paths.

Itr.	p	$\mathcal{O}$
1	–	A
2	A	AB, AC, AD
3	AB	AC, AD
4	AC	AD
5	AD	



- IDDFS is like performing DFS multiple times on a sub-tree of possible paths.

Itr.	p	$\mathcal{O}$
1	–	A
2	A	AB, AC, AD
3	AB	ABC, AC, AD
4	ABC	AC, AD
5	AC	ACB, ACD, ACE, AD
6	ACB	ACD, ACE, AD



- **Complete** for $b < \infty$.
  - Each IDDFS iteration is a DFS with $m = d_{\max}$.
  - Since $d_{\max}$ is finite, each iteration of IDDFS is complete.
- **Suboptimal**.
  - Action costs are not used by the algorithm.
- **Time Complexity**: $O(b^d)$.
  - Each IDDFS iteration is a DFS with $m = d_{\max}$.
  - Thus, each iteration of IDDFS has a time-complexity of $O(b^{d_{\max}})$.
- **Space Complexity**: $O(bd)$.
  - Each IDDFS iteration is a DFS with $m = d_{\max}$.
  - Thus, each iteration of IDDFS has a space-complexity of $O(bd_{\max})$.

- Below, we summarize the properties of BFS, DFS, UCS, and IDDFS:

Property	BFS	DFS	UCS	IDDFS
Complete	$b < \infty$	$b, m < \infty$	$b < \infty$	$b < \infty$
Optimal	constant c	never	always	never
Time Complexity	$O(b^{d+1})$	$O(b^m)$	$O(b^{c^*/\varepsilon+1})$	$O(b^d)$
Space Complexity	$O(b^{d+1})$	$O(bm)$	$O(b^{c^*/\varepsilon+1})$	$O(bd)$

- IDDFS seems to combine the best aspects of BFS and DFS. However, it is still not an optimal algorithm, and thus cannot be used with non-uniform action costs

- All of the algorithms discussed thus far work by maintaining a list of discovered paths (i.e., $\mathcal{O}$), and iteratively explore them to find a goal:
  - BFS explores shorter paths first
  - DFS explores longer paths first
  - UCS explores cheaper paths first
- In all cases, the order depends only on the length or cost of the path.
- In practice, the actual states traversed by the paths can provide substantial information in determining which paths are "more promising".
- Since the aforementioned algorithms do not make use of such information, we say that they are **uninformed**.
- Algorithms that do make use of this information are then **informed**.