Interim Review

Probability
Recall \( P(A, B, C, D) = P(A \land B \land C \land D) \)
Recall \( P(a, b, c, d) = P(A = a \land B = b \land C = c \land D = d) \)
Recall \( \sum_a P(a, b) = \sum_{a \in \text{Domain}(A)} P(A = a \land B = b) \)

Conditional Probability (Section 13.2.1, pp. 485, 492, 514)
Product Rule (= Chain Rule or Factoring) (Section 13.2.1; pp. 485, 495, 514)
Sum Rule (= Marginalization or Summing Out) (Section 13.3; p. 492)
Bayes Rule (Section 13.5)

Uninformed Search
Summary of algorithms (Section 3.4.7, Fig. 3.21)
Depth-first Search (DFS) and Friends (Sections 3.4.3-5)
Breadth-first Search (BFS) (Section 3.4.1)
Iterative Deepening Search (IDS) (Section 3.4.5)
(Please read lecture topic material before and after each lecture on that topic)
Conditional Probability

\[ P(A \mid B) = \frac{P(A, B)}{P(B)} \]

Area = Probability of Event

\[ P(A \cap B) = P(A) + P(B) - P(A \cup B) \]

Entire Sample Space: \( P(S) = 1 \)
Using the Product Rule

Recall \( P(a, b, c, d) = P(A = a \land B = b \land C = c \land D = d) \)

- **Applies to any number of variables:**
  - \( P(a, b, c) = P(a, b \mid c) \cdot P(c) = P(a \mid b, c) \cdot P(b, c) \)
  - \( P(a, b, c \mid d, e) = P(a \mid b, c, d, e) \cdot P(b, c \mid d, e) \)

- **Factoring:** (AKA Chain Rule for probabilities)
  - **By the product rule, we can always write:**
    \( P(a, b, c, \ldots z) = P(a \mid b, c, \ldots z) \cdot P(b, c, \ldots z) \)
  
  - **Repeatedly applying this idea, we can write:**
    \( P(a, b, c, \ldots z) = P(a \mid b, c, \ldots z) \cdot P(b \mid c, \ldots z) \cdot P(c \mid \ldots z) \cdots P(z) \)
  
  - This holds for any ordering of the variables
Using the Sum Rule
Recall \( P(a, b, c, d) = P(A = a \land B = b \land C = c \land D = d) \)
Recall \( \Sigma_a P(a, b) = \Sigma_{a \in \text{Domain}(A)} P(A = a \land B = b) \)

• We can marginalize variables out of any joint distribution by simply summing over that variable:
  – \( P(b) = \Sigma_a \Sigma_c \Sigma_d P(a, b, c, d) \)
  – \( P(a, d) = \Sigma_b \Sigma_c P(a, b, c, d) \)

• **For Example:** Determine probability of catching a fish
  – Given a set of probabilities \( P(\text{CatchFish, Day, Lake}) \)
  – Where:
    - \( \text{CatchFish} = \{\text{true, false}\} \)
    - \( \text{Day} = \{\text{mon, tues, wed, thurs, fri, sat, sun}\} \)
    - \( \text{Lake} = \{\text{buel lake, ralph lake, crystal lake}\} \)

  – Need to find \( P(\text{CatchFish} = \text{True}) \):
    - \( P(\text{CatchFish} = \text{true}) = \Sigma_{\text{day}} \Sigma_{\text{lake}} P(\text{CatchFish} = \text{true, day, lake}) \)
Derivation of Bayes’ Rule

Recall \( P(a, b, c, d) = P(A = a \land B = b \land C = c \land D = d) \)

• **Start from Product Rule:**
  
  \[ P(a, b) = P(a \mid b) P(b) = P(b \mid a) P(a) \]

• **Isolate Equality on Right Side:**
  
  \[ P(a \mid b) P(b) = P(b \mid a) P(a) \]

• **Divide through by P(b):**
  
  \[ P(a \mid b) = \frac{P(b \mid a) P(a)}{P(b)} \quad \text{ <-- Bayes’ Rule} \]
Summary of Probability Rules

Recall $P(A, B, C, D) = P(A \land B \land C \land D)$
Recall $P(a, b, c, d) = P(A = a \land B = b \land C = c \land D = d)$
Recall $\sum_a P(a, b) = \sum_{a \in \text{Domain}(A)} P(A = a \land B = b)$

• **Conditional Probability:**
  - $P(a \mid b) = \frac{P(a, b)}{P(b)}$

• **Product Rule:** (AKA chain Rule, Factoring)
  - $P(a, b) = P(a \mid b) P(b) = P(b \mid a) P(a)$
  - $P(a, b, c, d) = P(a \mid b, c, d) P(b \mid c, d) P(c \mid d) P(d)$

• **Sum Rule:** (AKA Law of Total Probability)
  - $P(a) = \sum_b P(a, b) = \sum_b P(a \mid b) P(b)$
    - where $B$ is any set of random variables
  - $P(a) = \sum_{b, c, d} P(a, b, c, d)$

• **Bayes’ Rule:**
  - $P(b \mid a) = \frac{P(a \mid b) P(b)}{P(a)}$
Interim Review

**Probability**

Recall $P(A, B, C, D) = P(A \land B \land C \land D)$
Recall $P(a, b, c, d) = P(A = a \land B = b \land C = c \land D = d)$
Recall $\sum_{a} P(a, b) = \sum_{a \in \text{Domain}(A)} P(A = a \land B = b)$

Conditional Probability (Section 13.2.1, pp. 485, 492, 514)
Product Rule (= Chain Rule or Factoring) (Section 13.2.1; pp. 485, 495, 514)
Sum Rule (= Marginalization or Summing Out) (Section 13.3; p. 492)
Bayes Rule (Section 13.5)

**Uninformed Search**

Summary of algorithms (Section 3.4.7, Fig. 3.21)
Depth-first Search and Friends (Sections 3.4.3-5)
Breadth-first Search (Section 3.4.1)

(Please read lecture topic material before and after each lecture on that topic)
# Summary of algorithms

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Uniform-Cost</th>
<th>Depth-First</th>
<th>Depth-Limited</th>
<th>Iterative Deepening DLS</th>
<th>Bidirectional (if applicable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes[a]</td>
<td>Yes[a,b]</td>
<td>No</td>
<td>No</td>
<td>Yes[a]</td>
<td>Yes[a,d]</td>
</tr>
<tr>
<td>Time</td>
<td>$O(b^d)$</td>
<td>$O(b^{1+C*/\varepsilon})$</td>
<td>$O(b^m)$</td>
<td>$O(b^l)$</td>
<td>$O(b^d)$</td>
<td>$O(b^{d/2})$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(b^d)$</td>
<td>$O(b^{1+C*/\varepsilon})$</td>
<td>$O(bm)$</td>
<td>$O(bl)$</td>
<td>$O(bd)$</td>
<td>$O(b^{d/2})$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes[c]</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes[c]</td>
<td>Yes[c,d]</td>
</tr>
</tbody>
</table>

There are a number of footnotes, caveats, and assumptions. See Fig. 3.21, p. 91.

[a] complete if $b$ is finite
[b] complete if step costs $\geq \varepsilon > 0$
[c] optimal if step costs are all identical
(d) if both directions use breadth-first search
(Also if both directions use uniform-cost search with step costs $\geq \varepsilon > 0$)

generally the preferred uninformed search strategy

Note that $d \leq \lfloor 1+C*/\varepsilon \rfloor$
Depth-first Search and Friends

Depth-limited search & IDS
Depth-first Search = Depth-Limited-Search with limit = ∞

function \textsc{Depth-Limited-Search}( \textit{problem}, \textit{limit}) \textbf{returns} soln/fail/cutoff
\textsc{Recursive-DLS}(\textsc{Make-Node}(\textsc{Initial-State}[\textit{problem}]), \textit{problem}, \textit{limit})

function \textsc{Recursive-DLS}(\textit{node}, \textit{problem}, \textit{limit}) \textbf{returns} soln/fail/cutoff
\texttt{cutoff-occurred?} \leftarrow \texttt{false}
\textbf{if} \textsc{Goal-Test}[\textit{problem}](\textsc{State}[\textit{node}]) \textbf{then return} \textsc{Solution}(\textit{node})
\textbf{else if} \textsc{Depth}[\textit{node}] = \textit{limit} \textbf{then return} \textsc{cutoff}
\textbf{else for each} \textit{successor} in \textsc{Expand}(\textit{node}, \textit{problem}) \textbf{do}
\texttt{result} \leftarrow \textsc{Recursive-DLS}(\textit{successor}, \textit{problem}, \textit{limit})
\textbf{if} \texttt{result} = \textsc{cutoff} \textbf{then} \texttt{cutoff-occurred?} \leftarrow \texttt{true}
\textbf{else if} \texttt{result} \neq \textsc{failure} \textbf{then return} \texttt{result}
\textbf{if} \texttt{cutoff-occurred?} \textbf{then return} \textsc{cutoff} \textbf{else return} \textsc{failure}

\begin{boxedtext}
function \textsc{Iterative-Deepening-Search}( \textit{problem}) \textbf{returns} a solution, or failure
\begin{itemize}
\item \textbf{inputs:} \textit{problem}, a problem
\item \textbf{for} \texttt{depth} \leftarrow 0 \textbf{to} ∞ \textbf{do}
\item \texttt{result} \leftarrow \textsc{Depth-Limited-Search}(\textit{problem}, \textit{depth})
\item \textbf{if} \texttt{result} \neq \textsc{cutoff} \textbf{then return} \texttt{result}
\end{itemize}
\end{boxedtext}
Properties of depth-first search

• **Complete?** No: fails in loops/infinite-depth spaces

• **Time?** $O(b^m)$ with $m =$maximum depth of space

• **Space?** $O(bm)$, i.e., linear space!

• **Optimal?** No: It may find a non-optimal goal first
Depth-first search (tree search)

Example hand-simulated search: Children returned in left-to-right order.

Order of node expansion: ____________________________
Path found: ____________________ Cost of path found: _____

White Node = Frontier
Gray Node = Expanded
Black Node = Deleted
Depth-first search (tree search)

Example hand-simulated search: Children returned in left-to-right order.

Order of node expansion: ____________
Path found: ____________ Cost of path found: ______

White Node = Frontier
Gray Node = Expanded
Black Node = Deleted
Depth-first search (tree search)

Example hand-simulated search: Children returned in left-to-right order.

Order of node expansion: S B
Path found: S B
Cost of path found: 5

White Node = Frontier
Gray Node = Expanded
Black Node = Deleted
Depth-first search (tree search)

Example hand-simulated search: Children returned in left-to-right order.

Order of node expansion: S B D
Path found: ____________  Cost of path found: ______
White Node = Frontier
Gray Node = Expanded
Black Node = Deleted
Depth-first search (tree search)

Example hand-simulated search: Children returned in left-to-right order.

Order of node expansion:  S B D F G
Path found:  S B D F G  Cost of path found:  9

White Node = Frontier
Gray Node = Expanded
Black Node = Deleted

Goal test before push

TECHNICAL NOTE: Technically, the goal node is not expanded, because no children of a goal node are generated. The goal node is listed in “Order of node expansion” for your convenience. Your answer is correct if you do not show the goal node in “Order of node expansion” — but it is a nicety to do so. Nevertheless, “Path found” *always* must show the goal node, because a path to a goal always must end in a goal.
Breadth-first graph search

function BREADTH-FIRST-SEARCH (problem) returns a solution, or failure

    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0 if
    problem.GOAL-TEST (node. STATE) then return SOLUTION (node) frontier ←
    a FIFO queue with node as the only element
    explored ← an empty set

    loop do
        if EMPTY? (frontier) then return failure
        node ← POP (frontier) /* chooses the shallowest node in frontier */
        add node. STATE to explored
        for each action in problem.ACTIONS (node. STATE) do
            child ← CHILD-NODE (problem, node, action)
            if child. STATE is not in explored or frontier then
                if problem.GOAL-TEST (child. STATE) then return SOLUTION (child)
                frontier ← INSERT (child, frontier)

Figure 3.11   Breadth-first search on a graph.
Properties of breadth-first search

- **Complete?** Yes, it always reaches a goal (if $b$ is finite)
- **Time?** $1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$
- **Space?** $O(b^d)$ (keeps every node in memory).
- **Optimal?** No, for general cost functions.
  Yes, if cost is a non-decreasing function only of depth.
Breadth-first search (tree search)

Example hand-simulated search: Children returned in left-to-right order.

Order of node expansion: __________________________
Path found: __________________ Cost of path found: _____
Breadth-first search (tree search)

Example hand-simulated search: Children returned in left-to-right order.

Order of node expansion: ________________
Path found: __________________ Cost of path found: ______

White Node = Frontier
Gray Node = Expanded
Black Node = Deleted
Breadth-first search (tree search)

Example hand-simulated search: Children returned in left-to-right order.

Order of node expansion: S B
Path found: S B C A Cost of path found: 7

White Node = Frontier
Gray Node = Expanded
Black Node = Deleted
Breadth-first search (tree search)

Example hand-simulated search: Children returned in left-to-right order.

Order of node expansion: S B A

Path found: S B A Cost of path found: 5

White Node = Frontier
Gray Node = Expanded
Black Node = Deleted

Note C1, C2, to distinguish two different paths to node C.
Breadth-first search (tree search)

Example hand-simulated search: Children returned in left-to-right order.

Order of node expansion: S B A D
Path found: S B A D
Cost of path found: 10

White Node = Frontier
Gray Node = Expanded
Black Node = Deleted
Breadth-first search (tree search)

Example hand-simulated search: Children returned in left-to-right order.

Order of node expansion:  S B A D1 C1
Path found: __________________________ Cost of path found: _____

White Node = Frontier
Gray Node = Expanded
Black Node = Deleted
Example hand-simulated search: Children returned in left-to-right order.

Order of node expansion: S B A D1 C1 C2
Path found: ____________________________ Cost of path found: ______

White Node = Frontier
Gray Node = Expanded
Black Node = Deleted
Breadth-first search (tree search)

Example hand-simulated search: Children returned in left-to-right order.

Order of node expansion:  _S_ _B_ _A_ _D1_ _C1_ _C2_ _F_ _G_
Path found: _______ S B D F G ________
Cost of path found: ____9____

White Node = Frontier
Gray Node = Expanded
Black Node = Deleted

Goal test before push

TECHNICAL NOTE: Technically, the goal node is not expanded, because no children of a goal node are generated. The goal node is listed in “Order of node expansion” for your convenience. Your answer is correct if you do not show the goal node in “Order of node expansion” — but it is a nicety to do so. Nevertheless, “Path found” *always* must show the goal node, because a path to a goal always must end in a goal.
Iterative Deepening Search (IDS)

Generally, the preferred uninformed search, if you know nothing about your search space.

```
function DEPTH-LIMITED-SEARCH( problem, limit) returns soln/fail/cutoff
    Recursive-DLS(Make-Node(Initial-State[problem]), problem, limit)

function Recursive-DLS(node, problem, limit) returns soln/fail/cutoff
    cutoff-occurred? ← false
    if GOAL-TEST[problem](State[node]) then return SOLUTION(node)
    else if DEPTH[node] = limit then return cutoff
    else for each successor in EXPAND(node, problem) do
        result ← Recursive-DLS(successor, problem, limit)
        if result = cutoff then cutoff-occurred? ← true
        else if result ≠ failure then return result
    if cutoff-occurred? then return cutoff else return failure
```

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution, or failure

inputs: problem, a problem

for depth ← 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH( problem, depth)
    if result ≠ cutoff then return result
```
Properties of iterative deepening search

- **Complete?** Yes
- **Time?** $O(b^d)$
- **Space?** $O(bd)$
- **Optimal?** No, for general cost functions. Yes, if cost is a non-decreasing function only of depth.

Generally the preferred uninformed search strategy.
Iterative Deepening search (tree search)

Example hand-simulated search: Children returned in left-to-right order.

Order of node expansion: none
Path found: S  Cost of path found: g=0

Fringe is goal-tested but not expanded, i.e., no children generated
Iterative Deepening search (tree search)

Example hand-simulated search: Children returned in left-to-right order.

Order of node expansion: S
Path found: S
Cost of path found: g=0

White Node = Frontier
Gray Node = Expanded
Black Node = Deleted

depth = limit = 1

Fringe is goal-tested but not expanded, i.e., no children generated
Iterative Deepening search (tree search)

Example hand-simulated search: Children returned in left-to-right order.

Order of node expansion: **S B A**

Path found: ______________________ Cost of path found: ______

depth = limit = 2

White Node = Frontier
Gray Node = Expanded
Black Node = Deleted

Note C1, C2, to distinguish two different paths to node C.

Fringe is goal-tested but not expanded, i.e., no children generated.
Iterative Deepening search (tree search)

Example hand-simulated search: Children returned in left-to-right order.

Order of node expansion: S B A D1 C1 C2
Path found: __________________________
Cost of path found: _______

White Node = Frontier
Gray Node = Expanded
Black Node = Deleted

depth = limit = 3

Fringe is goal-tested but not expanded, i.e., no children generated
Iterative Deepening search (tree search)

Example hand-simulated search: Children returned in left-to-right order.

Order of node expansion:  _S_ B A D1 C1 C2 F G
Path found:         _S_ B D F G           Cost of path found:  __9__

White Node = Frontier
Gray Node = Expanded
Black Node = Deleted

TECHNICAL NOTE: Technically, the goal node is not expanded, because no children of a goal node are generated. The goal node is listed in “Order of node expansion” for your convenience. Your answer is correct if you do not show the goal node in “Order of node expansion” — but it is a nicety to do so. Nevertheless, “Path found” *always* must show the goal node, because a path to a goal always must end in a goal.
Interim Review

**Probability**

Recall $P(A, B, C, D) = P(A \land B \land C \land D)$
Recall $P(a, b, c, d) = P(A = a \land B = b \land C = c \land D = d)$
Recall $\Sigma_a P(a, b) = \Sigma_{a \in \text{Domain}(A)} P(A = a \land B = b)$

Conditional Probability (Section 13.2.1, pp. 485, 492, 514)
Product Rule (= Chain Rule or Factoring) (Section 13.2.1; pp. 485, 495, 514)
Sum Rule (= Marginalization or Summing Out) (Section 13.3; p. 492)
Bayes Rule (Section 13.5)

**Uninformed Search**

Summary of algorithms (Section 3.4.7, Fig. 3.21)
Depth-first Search (DFS) and Friends (Sections 3.4.3-5)
Breadth-first Search (BFS) (Section 3.4.1)
Iterative Deepening Search (IDS) (Section 3.4.5)

(Please read lecture topic material before and after each lecture on that topic)