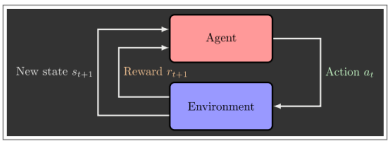
02 search problem

REMINDER: RATIONAL AGENTS



An **agent** is an entity that *perceives* and *acts*.

A **rational agent** selects actions that maximize its (expected) **utility**.

Characteristics of the **percepts**, **environment**, and **action space** dictate techniques for selecting rational actions.

RATIONAL AGENTS

Are rational agents omniscient?

- No - they are limited by what they can perceive

Are rational agents clairvoyant?

- No - they lack knowledge of environment dynamics

Do rational agents explore and learn?

- Yes - essential qualities required in unknown environments

So, rational agents are not necessarily successful, but they are autonomous.

ENVIRONMENT CATEGORIZATION

	Pacman	Robotaxi
Fully or Partially Observable	fully	partial
Single or Multi Agent	multi	multi
Deterministic or Stochastic	deterministic	stochastic
Static or Dynamic	static	dynamic
Discrete or Continuous	discrete	continuous

WHAT ARE SEARCH PROBLEMS?

A search problem consists of:

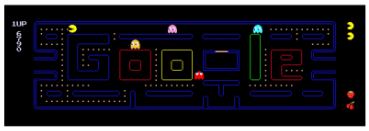
- A state space:

- A successor function (actions, costs):
- A start state and goal test

A solution is a sequence of actions (a plan) which transforms the start state to the goal test.

WHAT IS IN A STATE SPACE?

World State: Includes every last detail of the environment



Search State: Keeps only details necessary for planning

Problem: Pathing	Problem: Eat-All-Dots
States: (x,y) position	States: (x,y), boolean for each dot
Actions: NEWS	Actions: NEWS
Successor: update location	Successor: update location, boolean for dots
Goal Test: Is (x,y) == END	Goal Test: All dot booleans are false

03 uninformed-search

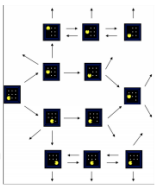
STATE SPACE GRAPHS

State Space Graph: A mathematical representation of a search problem

- Nodes are (abstracted) world configurations
- Arcs represent successors (action results)
- The goal test is a set of goal nodes (maybe only one)

In a state space graph, each state occurs only once

We can rarely build this full graph in memory (too big), but it is a useful idea



Tiny state space graph for a tiny search problem

TREE SEARCH ALGORITHM

Tree Search

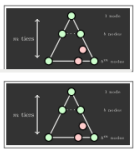
```
input  : problem, strategy
output: solution or failure
initialize the search tree using initial state of problem;
while forever do
    if there are no candidates for expansion then return failure;
    choose a leaf node for expansion according to strategy;
    if node contains a goal state then
        return the corresponding solution;
    else
        expand the node and add the resulting nodes to the
          search tree;
    end
end
```

Important Ideas:

- Fringe
- Expansion
- Exploration Strategy

Main question: which fringe nodes to explore?

BREADTH-FIRST SEARCH



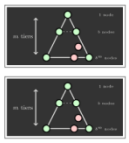
Strategy: expand the deepest node first

Implementation: fringe is a FILO or LIFO stack

DEPTH-FIRST SEARCH (DFS) PROPERTIES

What nodes DFS expand?

- Some left prefix of the tree.
- Could process the whole tree!
- If m is finite, takes time $\mathcal{O}(b^m)$



Is it complete?

- m could be infinite, so only if we prevent cycles (more later)

How much space does the fringe take?

- Only has siblings on path to root, so $\mathcal{O}(bm)$.

Is it optimal?

- No, it finds the "leftmost" solution, regardless of depth or cost.

Strategy: expand the shallowest node first

Implementation: fringe is a FIFO queue

BREADTH-FIRST SEARCH (BFS) PROPERTIES

What nodes BFS expand?

- Processes all nodes above shallowest solution
- Let depth of shallowest solution be s
- Search takes time $\mathcal{O}(b^s)$

ITERATIVE DEEPENING

Idea: get the space advantage of DFS with the time / shallow-solution advantage of BFS.

- Run a DFS with depth limit 1. If no solution ...
- Run a DFS with depth limit 2. If no solution ...
- Run a DFS with depth limit 3. If no solution ...

Is this not wastefully redundant?

- Generally most work happens in the deepest level searched, so it is not so bad.

Strategy: expand a cheapest node first

Implementation: fringe is a priority queue (priority: cumulative cost)

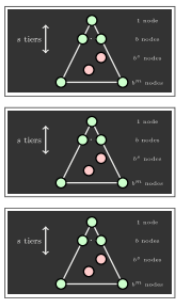
UCS PROPERTIES

What nodes UCS expand?

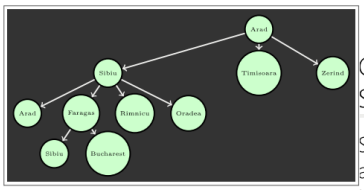
- Processes all nodes with cost less than cheapest solution!
- If that solution costs C^* and arcs cost at least ϵ , then the "effective depth" is roughly $\left(\frac{C^*}{\epsilon}\right)$
- Takes time $\mathcal{O}\left(b^{\frac{C^*}{\epsilon}}\right)$ (exponential in effective depth)

How much space does the fringe take?

- Has roughly the last tier, so $\mathcal{O}\left(b^{\frac{C^*}{\epsilon}}\right)$



04 informed search



GREEDY SEARCH

Strategy: expand a node that you think is closest to a goal state

What can go wrong?

- Heuristic: distance estimate to nearest goal for each state

A common case:

- Best-first takes you straight to the (wrong) goal

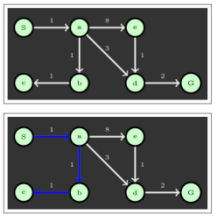
Worst-case: like a badly-guided DFS

COMBINING UCS AND GREEDY

Uniform-cost orders by path cost, or backward cost $g(n)$.

Greedy orders by goal proximity, or forward cost $h(n)$.

A^* Search orders by the sum: $f(n) = g(n) + h(n)$



OPTIMALITY OF A^* TREE SEARCH

Proof:

Imagine B is on the fringe

Some ancestor n of A is on the fringe too

Claim: n will be expanded before B

- $f(n)$ is less or equal to $f(A)$
- $f(A)$ is less than $f(B)$
- n expands before B

All ancestors of A expand before B

A expands before B

A^* search is optimal

$f(n) = g(n) + h(n)$

$f(n) \leq g(A)$ admissi

$g(A) = f(A)$ $h = 0$

$g(A) \leq g(B)$ B is su

$f(A) \leq f(B)$ $h = 0$ a

$f(n) \leq f(A) < f(B)$

ADMISSIBLE HEURISTICS

A heuristic h is admissible (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal

05 constraint satisfaction

WHAT ARE CONSTRAINT SATISFACTION PROBLEMS?

Special kind of search problems.

- N variables
- Values from domain D
- assignment satisfies constraints

States: partial assignment

Goal Test: satisfies all constraints

Successor Function: assign an unassigned variable

WHAT IS SEARCH FOR?

Given assumptions about the world: a single agent, deterministic actions, fully observed state, discrete state space

- Planning: sequences of actions
- The path to the goal is the important thing
- Paths have various costs, depths
- Heuristics give problem-specific guidance
- Identification: assignments to variables
- The goal itself is important, not the path
- All paths at the same depth (for some formulations)
- CSPs are a specialized class of identification problems

EXAMPLE: MAP COLORING

Variables: WA, NT, Q, NSW, V, SA, T

Domains: $D = \{red, green, blue\}$

Constraints: adjacent regions must have different colors

Implicit: $WA \neq NT$

Explicit: $(WA, NT) \in (red, green), (red, blue), \ldots$

Solutions: Assignments that satisfy all constraints, e.g.:

{WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green}

BACKTRACKING SEARCH

Backtracking search is the basic uninformed algorithm for solving CSPs

Idea 1: One variable at a time

- Variable assignments are commutative, so fix ordering
- Problem is commutative if order of application of any given set of actions has no effect on outcome.
- i.e., [WA = red then NT = green] same as [NT = green then WA = red]
- Only need to consider assignments to a single variable at each step

Idea 2: Check constraints as you go

- i.e. consider only values which do not conflict with previous assignments
- Might have to do some computation to check the constraints
- "Incremental goal test"

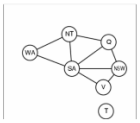
DFS with these two improvements is called backtracking search (not the best name)

CONSTRAINT GRAPHS

Binary CSP: each constraint relates two variables

Constraint Graph: nodes are variables, arcs are constraints

General-purpose CSP algorithms use the graph structure to speed up search. E.g., Tasmania is an independent subproblem.



STANDARD SEARCH FORMULATION

Standard search formulation of CSPs

States defined by the values assigned so far (partial assignments)

- Initial state: the empty assignment, {}
- Successor function: assign a value to an unassigned variable
- Goal test: the current assignment is complete and satisfies all constraints

Start with straightforward search and then improve.

BACKTRACKING SEARCH

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)
function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment given CONSTRAINTS[csp] then
      add {var = value} to assignment
      result ← RECURSIVE-BACKTRACKING(assignment, csp)
      if result ≠ failure then return result
      remove {var = value} from assignment
  return failure
```

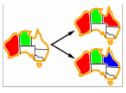
- Backtracking = DFS + variable-ordering + fail-on-violation

ORDERING: MINIMUM REMAINING VALUES

Variable Ordering: Minimum remaining values (MRV):

- Choose the variable with the fewest legal values left in its domain

ORDERING: LEAST CONSTRAINING VALUE



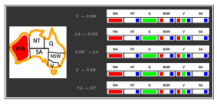
Value Ordering: Least Constraining Value

- Given a choice of variable, choose the least constraining value
- i.e., the one that rules out the fewest values in the remaining variables

ENFORCING ARC CONSISTENCY IN A CSP

ARC CONSISTENCY OF AN ENTIRE CSP

A simple form of propagation makes sure all arcs are consistent:



Important: If X loses a value, neighbors of X need to be rechecked.

Arc consistency detects failure earlier than forward checking

Can be run as a preprocessor or after each assignment

```
function AC-3(csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
  local variables: queue, a queue of arcs, initially all the arcs in csp
  while queue is not empty do
    (Xi, Xj) ← REMOVE-FIRST(queue)
    if REMOVE-INCONSISTENT-VALUES(Xi, Xj) then
      for each Xk in NEIGHBORS[Xi] do
        add (Xk, Xi) to queue

function REMOVE-INCONSISTENT-VALUES(Xi, Xj) returns true iff succeeds
  removed ← false
  for each x in DOMAIN[Xi] do
    if no value y in DOMAIN[Xj] allows (x,y) to satisfy the constraint Xi ↔ Xj
      then delete x from DOMAIN[Xi]; removed ← true
  return removed
```

Runtime: $\mathcal{O}(n^2d^3)$, can be reduced to $\mathcal{O}(n^2d^2)$

But detecting all possible future problems is NP-hard.

PROBLEM STRUCTURE

Real world problem can be decomposed into many subproblems.

Extreme case: independent subproblems

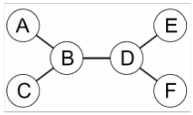
- Tasmania and mainland are independent subproblems

Independent subproblems are identifiable as connected components of constraint graph

Suppose a graph of n variables can be broken into subproblems of only c variables:

- Worst-case solution cost is $\mathcal{O}((n/c)(d^c))$, linear in n
- E.g., $n = 80$, $d = 2$, $c = 20$
- $2^{80} = 4$ billion years at 10 million nodes/sec
- $(4)(2^{20}) = 0.4$ seconds at 10 million nodes/sec

PROBLEM STRUCTURE



Theorem: if the constraint graph has no loops, the CSP can be solved in $\mathcal{O}(nd^2)$ time

Compare to general CSPs, where worst-case time is $\mathcal{O}(d^n)$

This property also applies to probabilistic reasoning (later): an example of the relation between syntactic restrictions and the complexity of reasoning.

PROBLEM STRUCTURE

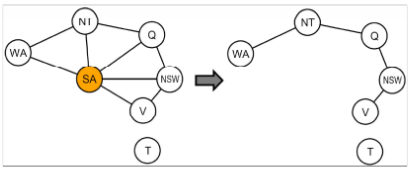
Algorithm for tree-structured CSPs:

- Order: Choose a root variable, order variables so that parents precede children

- Remove backward: For $i = n : 2$, apply RemoveInconsistent(Parent(X_i),X_i)
- Assign forward: For $i = 1 : n$, assign X_i consistently with Parent(X_i)

Runtime: $\mathcal{O}(nd^2)$

PROBLEM STRUCTURE



Conditioning: instantiate a variable, prune its neighbors' domains

Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree

Cutset size c gives runtime $\mathcal{O}((d^c)(n-c)d^2)$, very fast for small c.

07 adversarial search

DETERMINISTIC GAMES

Many possible formalizations, one is:

- **States** S (start at s_0)
- **Players** $P = \{1, \ldots, N\}$ (usually take turns)
- **Actions** A (may depend on player / state)
- **Transition Function** $S \times A \to S$
- **Terminal Test** $S \to \{t, f\}$
- **Terminal Utilities** $S \times P \to R$

Solution for a player is a policy: $S \to A$

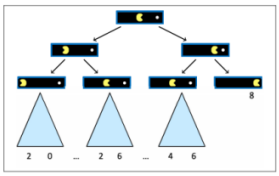
GAME THEORY

Zero-Sum Games

- Agents have opposite utilities (values on outcomes)
- Lets us think of a single value that one maximizes and the other minimizes
- Adversarial, pure competition

General Games

- Agents have independent utilities (values on outcomes)
- Cooperation, indifference, competition, and more are all possible
- More later on non-zero-sum games

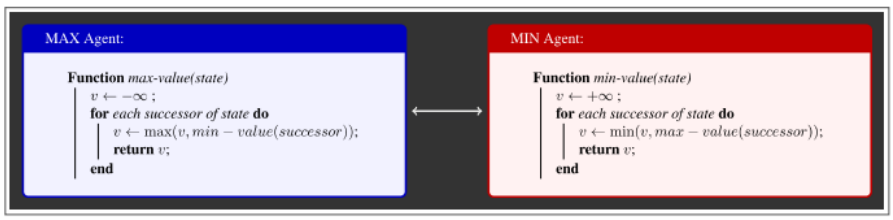


Value of State: The best achievable outcome (utility) from that state

Terminal States: $V(s)$ known

Non-Terminal States: $V(s) = \max_{s' \in children(s)} V(s')$

MINIMAX IMPLEMENTATION



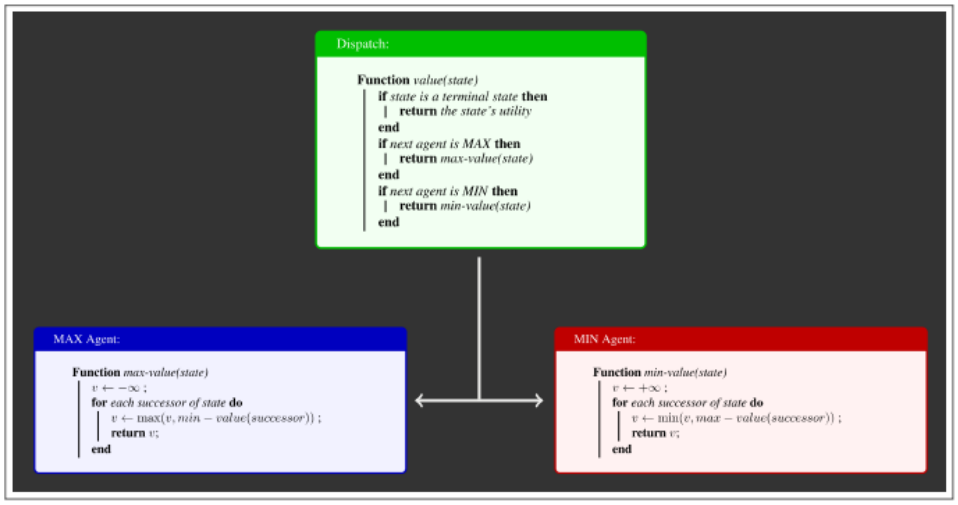
States Under Agent's Control: $V(s) = \max_{s' \in successors(s)} V(s')$

States Under Opponent's Control: $V(s') = \min_{s \in successors(s')} V(s)$

Terminal States: $V(s)$ known

MINIMAX VALUES

MINIMAX IMPLEMENTATION (DISPATCH)



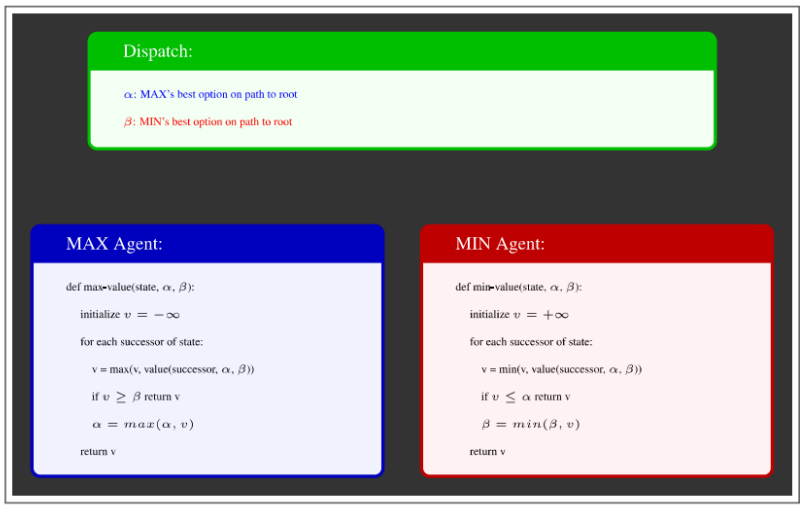
MINIMAX EFFICIENCY

How efficient is minimax?

- Just like (exhaustive) DFS
- Time: $\mathcal{O}(b^m)$
- Space: $\mathcal{O}(bm)$

Example: For chess, $b \approx 35$, $m \approx 100$

$A - B$ IMPLEMENTATION



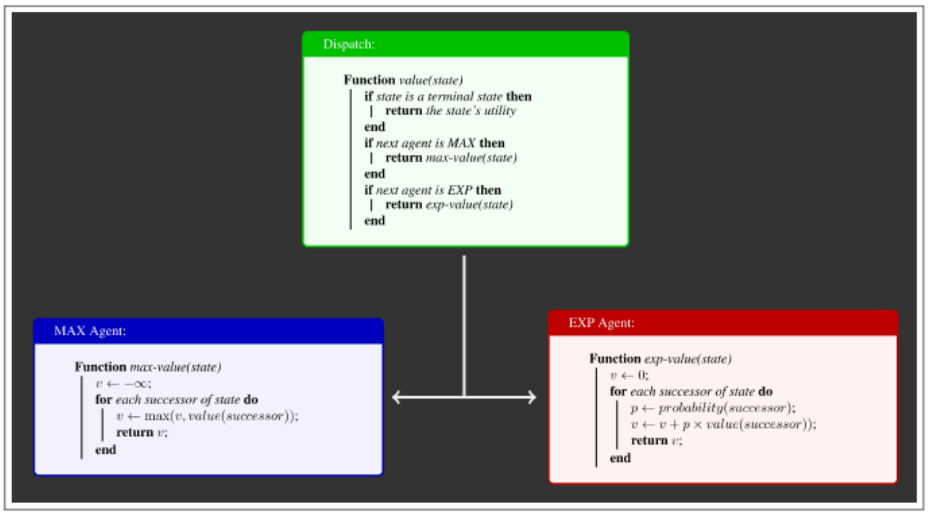
$A - B$ PRUNING

General configuration (MIN version)

- We are computing the $MIN - VALUE$ at some node n
- We are looping over n's children
- n's estimate of the childrens' min is dropping
- Who cares about n's value? MAX
- Let α be the best value that MAX can get at any choice point along the current path from the root.
- If n becomes worse than α, MAX will avoid it, so we can stop considering n's other children (it is already bad enough that it won't be played)

MAX version is symmetric

EXPECTIMAX IMPLEMENTATION



$A - B$ PRUNING PROPERTIES

This pruning has no effect on minimax value computed for the root.

Values of intermediate nodes might be wrong

- Important: children of the root may have the wrong value
- So the most naïve version won't let you do action selection

Good child ordering improves effectiveness of pruning

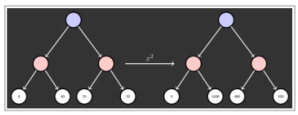
With "perfect ordering":

- Time complexity drops to $\mathcal{O}(b^{\frac{m}{2}})$
- Doubles solvable depth
- Full search of, e.g. chess, is still hopeless . . .

This is a simple example of metareasoning (computing about what to compute)

08

WHAT UTILITIES TO USE?



For worst-case minimax reasoning, terminal function scale does not matter

- We just want better states to have higher evaluations (get the ordering right)
- We call this insensitivity to monotonic transformations.

For average-case expectimax reasoning, we need magnitudes to be meaningful

09 linear programming

OPTIMIZATION FORMULATION

Diet Problem

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c}^T\mathbf{x} \\ s.t. \quad & a_{1,1}x_1 + a_{1,2}x_2 \leq b_1 \\ & a_{2,1}x_1 + a_{2,2}x_2 \leq b_2 \\ & a_{3,1}x_1 + a_{3,2}x_2 \leq b_3 \\ & a_{4,1}x_1 + a_{4,2}x_2 \leq b_4 \end{aligned}$$

$$A = \begin{bmatrix} -100 & -50 \\ 100 & 50 \\ 3 & 4 \\ -20 & -70 \end{bmatrix} \quad b = \begin{bmatrix} -2000 \\ 2500 \\ 100 \\ -700 \end{bmatrix} \quad \text{and } c = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}$$

LINEAR PROGRAMMING

Linear objective with linear constraints

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c}^T\mathbf{x} \\ s.t. \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \end{aligned}$$

As opposed to general optimization

$$\begin{aligned} \min_{\mathbf{x}} \quad & f_0(\mathbf{x}) \\ s.t. \quad & f_i(\mathbf{x}) \leq \mathbf{0}, i = 1, \ldots, M \\ & \mathbf{a}_i^T\mathbf{x} = \mathbf{b}_i, i = 1, \ldots, P \end{aligned}$$

SHAPES IN HIGHER DIMENSIONS

How do these linear shapes extend to 3-D, N-D?

	2-D	3-D	N-D
$a_1x_1 + a_2x_2 = b_1$	line	plane	hype
$a_1x_1 + a_2x_2 \leq b_1$	halfplane	halfspace	halfs
$a_{1,1}x_1 + a_{1,2}x_2 \leq b_1$ $a_{2,1}x_1 + a_{2,2}x_2 \leq b_2$ $a_{3,1}x_1 + a_{3,2}x_2 \leq b_3$ $a_{4,1}x_1 + a_{4,2}x_2 \leq b_4$	polygon	polyhedron	poly

10

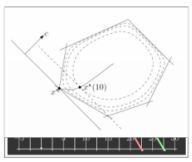
SOLVING AN IP

Branch and Bound algorithm

- Start with LP-relaxed version of IP
- If solution $\mathbf{x}^*_{LP}$ has non-integer value at $\mathbf{x}_i$
- Consider two branches with two different slightly more constrained LP problems:
- Left branch: Add constraint $\mathbf{x}_i \leq \boldsymbol{floor}(\mathbf{x}^*_{LP})$
- Right branch: Add constraint $\mathbf{x}_i \geq \boldsymbol{ceil}(\mathbf{x}^*_{LP})$
- Recursion. Stop going deeper:
- When the LP returns a worse objective than the best feasible IP objective you have seen before.
- When you hit an integer result from the LP
- When LP is infeasible

SOLVING A LP

Solutions are at feasible intersections of constraint boundaries.



Algorithms

- Check objective at all feasible intersections.
- Simplex
- Interior Point

SOLUTION OF LINEAR VS INTEGER PROGRAM

Let $\boldsymbol{y^*_{IP}}$ be the optimal objective of an integer program $\boldsymbol{P}$.

Let $\mathbf{x}^*_{IP}$ be the optimal point of an integer program $\boldsymbol{P}$.

Let $\boldsymbol{y^*_{LP}}$ be the optimal objective of the LP-relaxed version of $\boldsymbol{P}$.

Let $\mathbf{x}^*_{LP}$ be the optimal point of the LP-relaxed version of $\boldsymbol{P}$.

$$y^*_{IP} = \min_{\mathbf{x}} \mathbf{c}^T\mathbf{x} \quad s.t. \quad \mathbf{A}\mathbf{x} \leq \mathbf{b}, \ \mathbf{x} \in \mathbb{Z}^N$$

$$y^*_{LP} = \min_{\mathbf{x}} \mathbf{c}^T\mathbf{x} \quad s.t. \quad \mathbf{A}\mathbf{x} \leq \mathbf{b}$$

11 optimization

OPTIMIZATION PROBLEM: DEFINITION

$$\begin{aligned} \min_{\mathbf{x}} \quad & \boldsymbol{f(\mathbf{x})} \\ \text{s.t.} \quad & \mathbf{x} \in \mathcal{F} \end{aligned} \tag{4}$$

Optimization objective $\boldsymbol{f : \mathcal{F} \to \mathbb{R}}$

- $\boldsymbol{f(\mathbf{x}) = 1}$: Feasibility problem
- Simple functions
- Linear function $\boldsymbol{f(\mathbf{x}) = \mathbf{a}^T\mathbf{x}}$
- Convex function (next lecture)
- Complicated functions
- Can be implicitly represented through an algorithm which takes $\mathbf{x} \in \mathcal{F}$ as input, and outputs a value

OPTIMIZATION PROBLEM: EXAMPLE

Example: Traveling Salesman Problem (TSP)

- Problem: $\boldsymbol{n}$ cities, distance from city $\boldsymbol{i}$ to city $\boldsymbol{j}$ is $\boldsymbol{d(i,j)}$, find a tour (a closed path that visits every city exactly once) with minimal total distance.
- Variable $\mathbf{x}$: ordered list of cities being visited
- $\boldsymbol{x_i}$ is the index of the $\boldsymbol{i}$-th city being visited
- Feasible set $\boldsymbol{\mathcal{F} = \{\mathbf{x} :}$ **"each city visited exactly once"**$\boldsymbol{\}}$

$$\mathcal{F} = \{\mathbf{x} : \mathbf{x} \in \{1, \ldots, n\}^n; \sum_k \mathbb{I}(x_k = i) = 1, \forall i \in \{1, \ldots, n\}\} \tag{7}$$

- Objective function $\boldsymbol{f(\mathbf{x}) =}$ **"total distance when following $\mathbf{x}$"**

$$f(\mathbf{x}) = d(x_n, x_1) + \sum_{k=1}^{n-1} d(x_k, x_{k+1}) \tag{8}$$

Example: 8-Queens Problem (Solution 1)

- Variable $\mathbf{x}$: location of the queen in each column
- x_i is the row index of the queen in i-th column
- Feasible set $\mathcal{F} = \{\mathbf{x}:$ "no queens in the row, col, diag"$\}$

$$\mathcal{F} = \{\mathbf{x} : \mathbf{x} \in \{1,\dots,8\}^8; x_i \neq x_j, |x_i - x_j| \neq |i-j|, \forall i,j \in \{1,\dots,$$

- Objective function $f(\mathbf{x}) = 1$ (dummy)

Example: 8-Queens Problem (Solution 2)

- Variable x_i, y_i: index of row and column of the i-th queen
- Feasible set $F =$ $\{x, y :$ "no queens in the row, col, diag"$\}$

$$F = \{x, y : x, y \in \{1,\dots,8\}^8; \sum_i \mathbb{I}(x_i = k) = 1, \forall k \in \{1,\dots,8\};$$

$$\sum_i \mathbb{I}(y_i = k) = 1, \forall k \in \{1,\dots,8\}; |x_i - x_j| \neq |y_i - y_j|, \forall i,j \in \{1,$$

- Objective function $f(x) = 1$ (dummy)

12 convex optimization

Convex Optimization Problem:

$$\min_{\mathbf{x}} \ f(\mathbf{x})$$
$$\text{s.t. } \mathbf{x} \in \mathcal{F}$$

A special class of optimization problem

An optimization problem whose optimization objective f is a convex function and feasible region $\mathcal{F}$ is a convex set.

CONVEX COMBINATION

A point between two points

Given $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, a convex combination of them is any point of the form $\mathbf{z} = \theta\mathbf{x} + (1-\theta)\mathbf{y}$ where $\theta \in [0,1]$.

When $\theta \in (0,1)$, $\mathbf{z}$ is called a strict convex combination of $\mathbf{x}$, $\mathbf{y}$.

CONVEX SETS

Conceptually: Any convex combination of two points in the set is also in the set

Mathematically: A set $\mathcal{F}$ is convex if $\forall x, y \in \mathcal{F}, \forall \theta \in [0,1]$,

$$z = \theta x + (1-\theta)y \in \mathcal{F}$$

CONVEX FUNCTION

Value in the middle point is lower than average value

Let $\mathcal{F}$ be a convex set. A function $f : \mathcal{F} \to \mathbb{R}$ is convex in $\mathcal{F}$ if $\forall x, y \in \mathcal{F}, \forall \theta \in [0,1]$,

$$f(\theta\mathbf{x} + (1-\theta)\mathbf{y}) \leq \theta f(\mathbf{x}) + (1-\theta) f(\mathbf{y})$$

If $\mathcal{F} = \mathbb{R}^n$, we simply say f is convex.

- If f is a twice differentiable function of one variable, f is convex on an interval $[a,b] \subseteq \mathbb{R}$ iff (if and only if) its second derivative $f''(x) \geq 0$ in $[a,b]$

CONVEX OPTIMIZATION: DEFINITION

If f is a twice continuously differentiable function of n variables, f is convex on $\mathcal{F}$ iff its Hessian matrix of second partial derivatives is positive semidefinite on the interior of $\mathcal{F}$.

$$H(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} & \cdots \\ \frac{\partial^2 f}{\partial x_1 x_2} & \frac{\partial^2 f}{\partial x_2^2} & \cdots \\ \vdots & \vdots & \ddots \\ \frac{\partial^2 f}{\partial x_1 x_n} & \frac{\partial^2 f}{\partial x_2 x_n} & \cdots \end{bmatrix}$$

H is positive semidefinite in S if $\forall \mathbf{x} \in S, \forall \mathbf{z} \in \mathbb{R}^n, \mathbf{z}^T H(\mathbf{x}) \mathbf{z} \geq 0$

H is positive semidefinite in $\mathbb{R}^n$ iff all eigenvalues of H are non-negative.

Alternatively, prove $\mathbf{z}^T H(\mathbf{x}) \mathbf{z} = \sum_i g_i^2(x,z)$

CONCAVITY AND CONVEXITY

Concave function

- A function f is concave if $-f$ is convex
- Let $\mathcal{F}$ be a convex set. A function $f : \mathcal{F} \to \mathbb{R}$ is concave in $\mathcal{F}$ if $\forall \mathbf{x}, \mathbf{y} \in \mathcal{F}, \forall \theta \in [0,1]$,

$$f(\theta\mathbf{x} + (1-\theta)\mathbf{y}) \geq \theta f(\mathbf{x}) + (1-\theta) f(\mathbf{y})$$

The following is a convex optimization problem if f is a concave function and $\mathcal{F}$ is a convex set

$$\max_{\mathbf{x}} \ f(\mathbf{x})$$
$$\text{s.t. } \mathbf{x} \in \mathcal{F}$$

CONVEX OPTIMIZATION: LOCAL OPTIMA=GLOBAL OPTIMA

$$\min_{\mathbf{x}} \ f(\mathbf{x})$$
$$\text{s.t. } \mathbf{x} \in \mathcal{F}$$

Given an optimization problem, a point $\mathbf{x} \in \mathbb{R}^n$ is globally optimal if $\mathbf{x} \in \mathcal{F}$ and $\forall \mathbf{y} \in \mathcal{F}$, $f(\mathbf{x}) \leq f(\mathbf{y})$

Given an optimization problem, a point $\mathbf{x} \in \mathbb{R}^n$ is locally optimal if $\mathbf{x} \in \mathcal{F}$ and $\exists R > 0$ such that $\forall \mathbf{y}:$ $\mathbf{y} \in \mathcal{F}$ and $\|\mathbf{x} - \mathbf{y}\|_2 \leq R$, $f(\mathbf{x}) \leq f(\mathbf{y})$

Theorem 1: For a convex optimization problem, all locally optimal points are globally optimal

CONVEX OPTIMIZATION: HOW TO SOLVE?

Gradient descent: iteratively update the value of $\mathbf{x}$

- A simple algorithm for unconstrained optimization $\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x})$

Gradient Descent:

Initialize $x \leftarrow x_0$;
repeat
  $x \leftarrow x - \alpha \bigtriangledown_x f(x)$;
until *convergence*;

- Variants
- How to choose $\mathbf{x}_0$, e.g., $\mathbf{x}_0 = \mathbf{0}$
- How to choose and update step-size α, e.g., trial and error, line-search methods etc.
- How to define "convergence", e.g., $\|\mathbf{x}^{i+1} - \mathbf{x}^i\| \leq \epsilon$

PROJECTED GRADIENT DESCENT

Iteratively update the value of $\mathbf{x}$ while ensuring $\mathbf{x} \in \mathcal{F}$

Gradient Descent:

Initialize $x \leftarrow x_0$;
repeat
  $x \leftarrow P_{\mathcal{F}}(x - \alpha \bigtriangledown_x f(x))$;
until *convergence*;

- $P_{\mathcal{F}}$ projects a point to the constraint set.
- Variant:
- How to choose $P_{\mathcal{F}}$, e.g., $P_{\mathcal{F}} = \underset{\mathbf{x}' \in \mathcal{F}}{\operatorname{argmin}} \|\mathbf{x} - \mathbf{x}'\|_2^2$

13-14 Markov Decision Process

An MDP is defined by:

- A set of states $s \in S$
- A set of actions $a \in A$
- A transition function $T(s, a, s')$
  - Probability that a from s leads to s', i.e., $P(s'|s, a)$
  - Also called the model or the dynamics
- A reward function $R(s, a, s')$
  - Sometimes just $R(s)$ or $R(s')$
- A start state, s_0
- Maybe a terminal state

MDPs are non-deterministic search problems

- One way to solve them is with expectimax search
- We will have a new tool soon

STATIONARY PREFERENCES

Assumption: if we assume stationary preferences:

$$[a_1, a_2, \ldots] \succ [b_1, b_2, \ldots]$$
$$\Updownarrow$$
$$[r, a_1, a_2, \ldots] \succ [r, b_1, b_2, \ldots]$$

Then: there are only two ways to define utilities

- Additive utility:

$$U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \ldots$$

- Discounted utility:

$$U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$$

Problem: What if the game lasts forever? Do we get infinite rewards?

Solutions:

- Finite horizon: (similar to depth-limited search)
  - Terminate episodes after a fixed T steps (e.g. life)
  - Gives non-stationary policies (π depends on time left)
- Discounting: use $0 < \gamma < 1$

$$U([r_0, \ldots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq \frac{R_{max}}{1-\gamma}$$

  - Smaller γ means smaller "horizon" – shorter term focus
- Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "overheated" for racing)

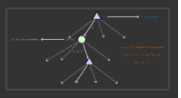
Value Iteration

Policy Iteration

- Policy Evaluation
- Policy Extraction

OPTIMAL QUANTITIES

The value (utility) of a state s: $V^*(s)$ = expected utility starting in s and acting optimally



The value (utility) of a q-state (s, a): $Q^*(s, a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

The optimal policy: $\pi^*(s)$ = optimal action from state s

VALUES OF STATES

Fundamental operation: compute the (expectimax) value of a state

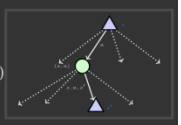
- Expected utility under optimal action
- Average sum of (discounted) rewards
- This is just what expectimax computed

Recursive definition of value:

$$V^*(s) = \max_a Q^*(s, a) \quad (1)$$

$$Q^*(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

$$V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')] \quad (2)$$

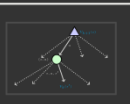


VALUE ITERATION

Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero

Given vector of $V_k(s)$ values, do one ply of expectimax from each state:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_k(s')]$$



Repeat until convergence

Complexity of each iteration: $O(S^2A)$

Theorem: will converge to unique optimal values

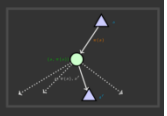
- Basic idea: approximations get refined towards optimal values
- Policy may converge long before values do

POLICY EVALUATION

How do we calculate the V's for a fixed policy π?

Idea 1: Turn recursive Bellman equations into updates (like value iteration)

$$V_0^\pi(s) = 0$$
$$V_{k+1}^\pi(s) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_k^\pi(s')]$$



Efficiency: $O(S^2)$ per iteration

Idea 2: Without the maxes, the Bellman equations are just a linear system

- Solve with Matlab (or your favorite linear system solver)

COMPUTING VALUES FROM ACTIONS

Imagine we have the optimal values $V^*(s)$

How should we act?

- It is not obvious

We need to do a mini-expectimax (one step)

$$\pi^*(s) = \operatorname{argmax}_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

This is called policy extraction, since it gets the policy implied by the values.

COMPUTING ACTIONS FROM Q-VALUES

Imagine we have the optimal q-values: $Q^*(s, a)$

How should we act?

- Completely trivial to decide !!

$$\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$$

Important lesson: actions are easier to select from q-values than values !!

POLICY ITERATION

Alternative approach for optimal values:

- Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities) until convergence
- Step 2: Policy improvement: update policy using one-step look-ahead with resulting converged (but not optimal) utilities as future values
- Repeat steps until policy converges

This is policy iteration

- It is still optimal !!
- Can converge (much) faster under some conditions

POLICY ITERATION

Evaluation: For fixed current policy π, find values with policy evaluation:

- Iterate until values converge:

$$V_{k+1}^{\pi_i}(s) = \sum_{s'} T(s, \pi_i(s), s')[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')]$$

Improvement: For fixed values, get a better policy using policy extraction

- One-step look-ahead:

$$\pi_{i+1}(s) = \operatorname{argmax}_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^{\pi_i}(s')]$$

15-17 reinforcement learning

MODEL-BASED LEARNING

Model-Based Idea:

- Learn an approximate model based on experiences
- Solve for values as if the learned model were correct

Step 1: Learn empirical MDP model

- Count outcomes s' for each s, a
- Normalize to give an estimate of $\hat{T}(s, a, s')$
- Discover each $\hat{R}(s, a, s')$ when we experience (s, a, s')

Step 2: Solve the learned MDP

- For example, use value iteration, as before

EXAMPLE: MODEL-BASED LEARNING

Input Policy π

	A	
B →	C →	D
	E ↑	

Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1
B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2
B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3
E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4
E, north, C, -1
C, east, A, -1
A, exit, x, -10

Learned Model

$\hat{T}(s, a, s')$

$T(B, east, C) = 1.00$
$T(C, east, D) = 0.75$
$T(C, east, A) = 0.25$

$\hat{R}(s, a, s')$

$R(B, east, C) = -1$
$R(C, east, D) = -1$
$R(D, exit, x) = +10$

Model-free learning

DIRECT EVALUATION

Goal: Compute values for each state under π

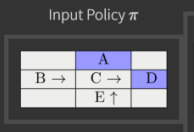
Idea: Average together observed sample values

- Act according to π
- Every time you visit a state, write down what the sum of discounted rewards turned out to be
- Average those samples

This is called direct evaluation

EXAMPLE: DIRECT EVALUATION

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Output Values

SAMPLE-BASED POLICY EVALUATION

We want to improve our estimate of V by computing these averages:

$$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^{\pi}(s')]$$

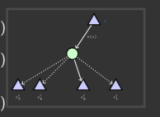
Idea: Take samples of outcomes s' (by doing the action) and average

$$sample_1 = R(s, \pi(s), s'_1) + \gamma V_k^{\pi}(s'_1) \quad (4)$$

$$sample_2 = R(s, \pi(s), s'_2) + \gamma V_k^{\pi}(s'_2) \quad (5)$$

$$\dots$$

$$sample_n = R(s, \pi(s), s'_n) + \gamma V_k^{\pi}(s'_n) \quad (6)$$



$$V_{k+1}^{\pi}(s) \leftarrow \frac{1}{n} \sum_i sample_i \quad (7)$$

16

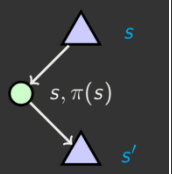
TEMPORAL-DIFFERENCE LEARNING

Big idea: learn from every experience

- Update $V(s)$ each time we experience a transition (s, a, s', r)
- Likely outcomes s' will contribute updates more often

Temporal difference learning of values

- Policy still fixed, still doing evaluation
- Move values toward value of whatever successor occurs: running average



Sample of $V(s)$: $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$ (1)

Update to $V(s)$: $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$ (2)

Same Update $V^{\pi}(s) = V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$ (3)

EXPONENTIAL MOVING AVERAGE

Exponential moving average

- The running interpolation update:

$$\bar{x}_n = (1-\alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$$

- Makes recent samples more important:

$$\bar{x}_n = \frac{x_n + (1-\alpha) \cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2 + \dots}$$

- Forgets about the past (distant past values were wrong anyway)

Decreasing learning rate (α) can give converging averages

EXAMPLE: TEMPORAL-DIFFERENCE LEARNING

Observed Transitions

States

Assume $\gamma = 1$, $\alpha = 0.5$

$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$

Active reinforcement learning

Q-LEARNING

Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma \max_{a'} Q_k(s', a')] \quad (1)$$

Learn $Q(s, a)$ values as you go

- Receive a sample (s, a, s', r)
- Consider your old estimate: $Q(s, a)$
- Consider your new sample estimate:

$$sample = R(s, a, s') + \gamma \max_{a'} Q(s', a') \quad (2)$$

- Incorporate the new estimate into a running average:

$$Q(s, a) \leftarrow (1-\alpha)Q(s, a) + (\alpha)sample \quad (3)$$

EXPLORATION FUNCTIONS

When to explore?

- Random actions: explore a fixed amount
- Better idea: explore areas whose badness is not (yet) established, eventually stop exploring

Exploration function

- Takes a value estimate u and a visit count n, and returns an optimistic utility, e.g. $f(u, n) = u + \frac{k}{n+1}$

Regular Q-Update : $Q(s, a) \leftarrow (1-\alpha)Q(s, a) + \alpha[R(s, a, s') + \gamma \max_{a'} Q(s', a')]$ (1)

Modified Q-Update : $Q(s, a) \leftarrow (1-\alpha)Q(s, a) + \alpha[R(s, a, s') + \gamma \max_{a'} f(Q(s', a'), N(s', a'))]$ (2)

- Note: this propagates the "bonus" back to states that lead to unknown states as well.

ch17

LINEAR VALUE FUNCTIONS

Using a feature representation, we can write a q function (or value function) for any state using a few weights:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

$$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$$

Advantage: our experience is summed up in a few powerful numbers

Disadvantage: states may share features but actually be very different in value.

APPROXIMATE Q-LEARNING

$$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$$

Q-learning with linear Q-functions:

$$transition = (s, a, r, s') \quad (1)$$

$$difference = [r + \gamma \max_{a'} Q(s', a')] - Q(s, a) \quad (2)$$

$$Q(s, a) \leftarrow Q(s, a) + \alpha[difference] \quad \text{ExactQ's} \quad (3)$$

$$w_i \leftarrow w_i + \alpha[difference] f_i(s, a) \quad \text{ApproximateQ's} \quad (4)$$

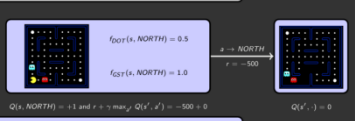
Intuitive interpretation:

- Adjust weights of active features
- E.g., if something unexpectedly bad happens, blame the features that were on: do not prefer all states with that state's features

Formal justification: online least squares

EXAMPLE Q-PACMAN

$$Q(s, a) = 4.0 f_{DOT}(s, a) - 1.0 f_{GST}(s, a)$$



$difference = -501 \longrightarrow$ $w_{DOT} \leftarrow 4.0 + \alpha[-501] \times 0.5$, $w_{GST} \leftarrow -1.0 + \alpha[-501] \times 1.0$

$$Q(s, a) = 3.0 f_{DOT}(s, a) - 3.0 f_{GST}(s, a)$$

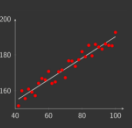
MINIMIZING ERROR

Imagine we had only one point x, with features f(x), target value y, and weights w:

$$error(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2 \quad (2)$$

$$\frac{\partial error(w)}{\partial w_m} = -\left(y - \sum_k w_k f_k(x)\right) f_m(x) \quad (3)$$

$$w_m \leftarrow w_m + \alpha\left(y - \sum_k w_k f_k(x)\right) f_m(x) \quad (4)$$



Approximate q update explained:

$$w_m \leftarrow w_m + \alpha[\underbrace{r + \gamma \max_a Q(s', a')}_{\text{"target"}} - \underbrace{Q(s, a)}_{\text{"prediction"}}] f_m(s, a)$$

18-Probability

MARGINAL DISTRIBUTIONS

Marginal distributions are sub-tables which eliminate variables

Marginalization (summing out): Combine collapsed rows by adding

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

T	P
hot	0.5
cold	0.5

W	P
sun	0.6
rain	0.4

CONDITIONAL DISTRIBUTIONS

Conditional distributions are probability distributions over some variables given fixed values of others

$P(W|T = hot)$

W	P
sun	0.8
rain	0.2

$P(W|T = cold)$

W	P
sun	0.4
rain	0.6

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

NORMALIZATION TRICKS

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

SELECT the joint probabilities matching the evidence →

T	W	P
cold	sun	0.2
cold	rain	0.3

NORMALIZE the selection (make it sum to one) →

W	P
sun	0.4
rain	0.6

19Bayesian Networks-representation

CONDITIONAL INDEPENDENCE

Unconditional (absolute) independence very rare (why?)

Conditional independence is our most basic and robust form of knowledge about uncertain environments.

X is conditionally independent of Y given Z i.e., $X \perp\!\!\!\perp Y|Z$

- iff:

$$\forall x, y, z: \ P(x, y|z) = P(x|z)P(y|z)$$

- or, equivalently, iff

$$\forall x, y, z: \ P(x|y, z) = P(x|z)$$

GRAPHICAL MODEL NOTATION

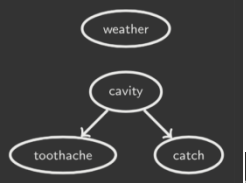
Nodes: variables (with domains)

- Can be assigned (observed) or unassigned (unobserved)

Arcs: interactions

- Similar to CSP constraints
- Indicate "direct influence" between variables
- Formally: encode conditional independence (more later)

For now: imagine that arrows mean direct causation (in general, they don't.)



PROBABILITIES IN BNS

Why are we guaranteed that setting the following results in a proper joint distribution?

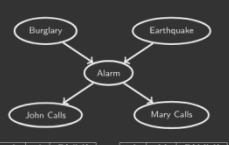
$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i|parents(x_i))$$

20bayesian-networks-independence

EXAMPLE: ALARM NETWORK

B	P(B)
+b	0.001
-b	0.999

E	P(E)
+e	0.002
-e	0.998



B	E	A	P(A|B,E)
+b	+e	+a	0.95
+b	+e	-a	0.05
+b	-e	+a	0.94
+b	-e	-a	0.96
-b	+e	+a	0.29
-b	+e	-a	0.71
-b	-e	+a	0.001
-b	-e	-a	0.999

A	J	P(J|A)
+a	+j	0.9
+a	-j	0.1
-a	+j	0.05
-a	-j	0.95

A	M	P(M|A)
+a	+m	0.7
+a	-m	0.3
-a	+m	0.01
-a	-m	0.99

$$P(+b, -e, +a, -j, +m) = P(+b)P(-e)P(+a|+b, -e)P(-j|+a)P(+m|+a)$$
$$= 0.001 \times 0.998 \times 0.94 \times 0.1 \times 0.7$$

How big is a joint distribution over N Boolean variables?

$$2^N$$

How big is an N-node net if nodes have up to k parents?

$$\mathcal{O}(N * 2^{k+1})$$

D-Separation

CAUSAL CHAINS

This configuration is a "causal chain"

$$P(x, y, z) = P(x)P(y|x)P(z|y)$$

Guaranteed X independent of Z given Y?

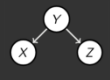
$$P(z|x, y) = \frac{P(x, y, z)}{P(x, y)}$$
$$= \frac{P(x)P(y|x)P(z|y)}{P(x)P(y|x)}$$
$$= P(z|y)$$

Evidence along the chain "blocks" the influence

COMMON CAUSE

This configuration is a "common cause"

- Y: project due
- X: forums busy
- Z: lab full



$$P(x, y, z) = P(y)P(x|y)P(z|y)$$

Guaranteed X independent of Z?

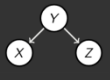
- One example set of CPTs for which X is not independent of Z is sufficient to show this independence is not guaranteed.
- Example:
  - Project due causes both forums busy and lab full
  - In numbers:

$$P(+x|+y) = 1, P(-x|-y) = 1$$
$$P(+z|+y) = 1, P(-z|-y) = 1$$

COMMON CAUSE

This configuration is a "common cause"

- Y: project due
- X: forums busy
- Z: lab full



$$P(x, y, z) = P(y)P(x|y)P(z|y)$$

Guaranteed X and Z independent given Y?

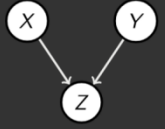
$$P(z|x, y) = \frac{P(x, y, z)}{P(x, y)}$$
$$= \frac{P(y)P(x|y)P(z|y)}{P(y)P(x|y)}$$
$$= P(z|y)$$

Observing the cause blocks influence between effects

COMMON EFFECT

Last configuration: two causes of one effect (v-structures)

- X: raining
- Y: ballgame
- Z: traffic



Are X and Y independent?

- Yes the ballgame and the rain cause traffic, but they are not correlated
- Still need to prove they must be (try it)

Are X and Y independent given Z?

- No seeing traffic puts the rain and the ballgame in competition as explanation

This is backwards from the other cases

- Observing an effect activates influence between possible causes.

ACTIVE/INACTIVE PATHS

Question: Are X and Y conditionally independent given evidence variables Z?

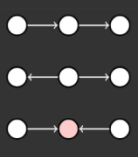
- Yes, if X and Y "d-separated" by Z
- Consider all (undirected) paths from X to Y
- No active paths = independence !!

A path is active if each triple is active:

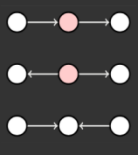
- Causal chain $A \rightarrow B \rightarrow C$ where B is unobserved (either direction)
- Common cause $A \leftarrow B \rightarrow C$ where B is unobserved
- Common effect (aka v-structure): $A \rightarrow B \leftarrow C$ where B or *one of its descendents* is observed

All it takes to block a path is a single inactive segment

Active Triples



Inactive Triples

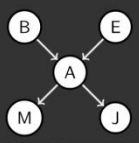


21bayesian-networks-inference

INFERENCE BY ENUMERATION IN BAYES NET

Given unlimited time, inference in BNs is easy

Reminder of inference by enumeration by example:



$$P(B|+j,+m) \propto P(B,+j,+m)$$
$$= \sum_{e,a} P(B,e,a,+j,+m)$$
$$= \sum_{e,a} P(B)P(e)P(a|B,e)P(+j|a)P(+m|a)$$

FACTORS I

Joint distribution: P(X,Y)

- Entries $P(x,y)$ for all x, y
- Sums to 1

Selected joint: $P(x,Y)$

- A slice of the joint distribution
- Entries $P(x,y)$ for fixed x, all y
- Sums to $P(x)$

Number of capitals = dimensionality of the table

$P(T,W)$

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

$P(cold,W)$

T	W	P
cold	sun	0.2
cold	rain	0.3

FACTORS II

Single conditional: $P(Y|x)$

- Entries $P(y|x)$ for fixed x, all y
- Sums to 1

Family of conditionals: $P(Y|X)$

- Multiple conditionals
- Entries $P(y|x)$ for all x, y
- Sums to $|X|$

$P(W|cold)$

T	W	P
cold	sun	0.4
cold	rain	0.6

$P(W|T)$

T	W	P
hot	sun	0.8
hot	rain	0.2
cold	sun	0.4
cold	rain	0.6

FACTORS III

Specified family: $P(y|X)$

- Entries $P(y|x)$ for fixed y, but for all x
- Sums to . . . who knows

$P(rain|T)$

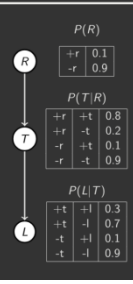
T	W	P
hot	rain	0.2
cold	rain	0.6

EXAMPLE: TRAFFIC DOMAIN

Random Variables

- R: raining
- T: traffic
- L: late for class

$$P(L) = ?$$
$$= \sum_{r,t} P(r,t,L)$$
$$= \sum_{r,t} P(r)P(t|r)P(L|t)$$



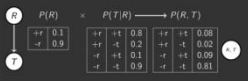
OPERATION 1: JOIN FACTORS

First basic operation: joining factors

Combining factors:

- Just like a database join
- Get all factors over the joining variable
- Build a new factor over the union of the variables involved

Example: Join on R



- Computation for each entry: pointwise products $\forall r,t : P(r,t) = P(r)P(t|r)$

OPERATION 2: ELIMINATE

Second basic operation: marginalization

Take a factor and sum out a variable

- Shrinks a factor to a smaller one
- A projection operation

Example:

$P(R,T)$

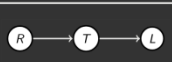
+r	+t	0.08
+r	-t	0.02
-r	+t	0.09
-r	-t	0.81

sum R →

$P(T)$

+t	0.17
-t	0.83

TRAFFIC DOMAIN



$P(L) = ?$

Inference by Enumeration

$$= \sum_t \sum_r P(L|t)P(r)P(t|r)$$

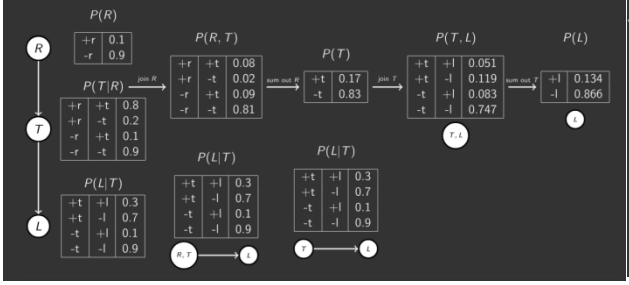
- join r
- join t
- eliminate r
- eliminate t

Variable Elimination

$$= \sum_t P(L|t) \sum_r P(r)P(t|r)$$

- join r
- eliminate r
- join t
- eliminate t

MARGINALIZING EARLY (VE)



GENERAL VARIABLE ELIMINATION

Query: $P(Q|E_1 = e_1, \ldots, E_k = e_k)$

Start with initial factors:

- Local CPTs (but instantiated by evidence)

While there are still hidden variables (not Q or evidence):

- Pick a hidden variable H
- Join all factors mentioning H
- Eliminate (sum out) H

Join all remaining factors and normalize

22bayesian-networks-sampling

PRIOR SAMPLING

Prior Sampling:

Function *prior-sampling(bayes net)*
  for $i = 1, 2, \ldots, n$ **do**
    Sample x_i from $P(X_i|Parents(X_i))$;
    return $(x_1, x_2, \ldots, x_n)$;
  end

PRIOR SAMPLING

This process generates samples with probability:

$$S_{PS}(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i|Parents(X_i)) = P(x_1, \ldots, x_n)$$

i.e., the BN's joint probability.

Let the number of samples of an event be $N_{PS}(x_1, \ldots, x_n)$

Then,

$$\lim_{N\to\infty} \hat{P}(x_1, \ldots, x_n) = \lim_{N\to\infty} \frac{N_{PS}(x_1, \ldots, x_n)}{N}$$
$$= S_{PS}(x_1, \ldots, x_n)$$
$$= P(x_1, \ldots, x_n)$$

i.e., the sampling procedure is consistent

REJECTION SAMPLING

Rejection Sampling:

Function *rejection-sampling(bayes-net, evidence-instantiation)*
  for $i = 1, 2, \ldots, n$ **do**
    Sample x_i from $P(X_i|Parents(X_i))$;
    if x_i *not consistent with evidence* **then**
      Reject: return – no sample is generated in this cycle
    end
    return $(x_1, x_2, \ldots, x_n)$;
  end

LIKELIHOOD WEIGHTING: ALGORITHM

Likelihood Weighting:

Function *likelihood-weighting(bayes-net, evidence-instantiation)*
- $w = 1.0$;
- **for** $i = 1, 2, \ldots, n$ **do**
  - Sample x_i from $P(X_i|Parents(X_i))$;
  - **if** x_i *is an evidence variable* **then**
    - X_i = *observation* x_i for X_i ;
    - Set $w = w * P(x_i|Parents(X_i))$;
  - **end**
  - **else**
    - Sample x_i from $P(X_i|Parents(X_i))$;
  - **end**
  - **return** $(x_1, x_2, \ldots, x_n)$, w;
- **end**

LIKELIHOOD WEIGHTING

Sampling distribution if z sampled and e fixed evidence

$$S_{WS}(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{l} P(z_i|Parents(Z_i))$$

Now, samples have weights

$$w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{m} P(e_i|Parents(E_i))$$

Together, weighted sampling distribution is consistent

$$S_{WS}(z, e) \cdot w(z, e) = \prod_{i=1}^{l} P(z_i|Parents(z_i)) \prod_{i=1}^{l} P(e_i|Parents(e_i)) = P(\mathbf{z}, \mathbf{e})$$

GIBBS SAMPLING EXAMPLE: $P(S|+R)$

Step 1: Fix evidence
- $R = +r$

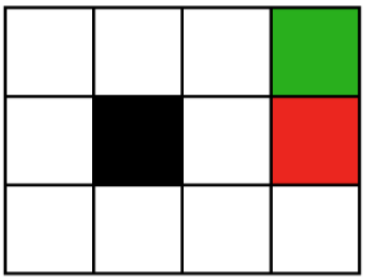
Step 2: Initialize other variables
- randomly

Step 3: Repeat
- Choose a non-evidence variable X
- Resample X from $P(X|\text{all other variables})$

Tutorial

1 MDP Exercise: Value Iteration

For today's class the environment we're exploring is a grid world, pictured below.



With this grid world, we define the following Markov decision process $\langle S, A, T, R, \gamma \rangle$:

- S: our states are represented as grid cells, each grid cell is one state in our world;
- A: the actions the robot can take in the world include moving *up*, *down*, *left*, and *right*; so there are 4 total actions that can be taken in this world;
- T: the robot moves as expected with action a from one state (s) to another (s') with probability 0.8. With probability 0.1, the robot moves to either left or right in perpendicular to action a. Here are some examples:
  - If a specifies that the robot should move to the *right*, with probability 0.8 the robot will move right. With probability 0.1 the robot will move up. And with probability 0.1 the robot will move down.
  - If a specifies that the robot should move to the *down*, with probability 0.8 the robot will move down. With probability 0.1 the robot will move right. And with probability 0.1 the robot will move left.
- R: Reward function only depends on the state of the world, i.e., $R(green) = +1$, $R(red) = -1$, and $R(\text{everywhere else}) = -2$.
- γ: for today's class we'll use a fixed discounting factor γ of value of 0.8.

Note: To best keep track of each state, we will denote each state by it's row and column where the bottom left is the origin $[0, 0]$. For example, the green state at time k can be represented as $s_k = [2, 3]$

	Trajectory 1	Trajectory 2	Trajectory 3	Trajectory 4	Trajectory 5	Trajectory 6
s_0	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$
V	$V([0, 0]) = -2$	$V([0, 0]) = -2.32$	$V([0, 0]) = -2.47$	$V([0, 0]) = -3.77$	$V([0, 0]) = -4.89$	$V([0, 0]) = -5.09$
a_0	UP	UP	RIGHT	RIGHT	UP	RIGHT
s_1	$[1, 0]$	$[1, 0]$	$[0, 1]$	$[0, 1]$	$[1, 0]$	$[0, 1]$
V	$V([1, 0]) = -2$	$V([1, 0]) = -3.6$	$V([0, 1]) = -2$	$V([0, 1]) = -3.6$	$V([1, 0]) = -4.88$	$V([0, 1]) = -4.07$
a_1	UP	UP	RIGHT	RIGHT	UP	RIGHT
s_2	$[2, 0]$	$[2, 0]$	$[0, 2]$	$[0, 2]$	$[2, 0]$	$[0, 2]$
V	$V([2, 0]) = -2$	$V([2, 0]) = -3.6$	$V([0, 2]) = -2$	$V([0, 2]) = -2.34$	$V([2, 0]) = -4.88$	$V([0, 2]) = -3.70$
a_2	RIGHT	RIGHT	UP	RIGHT	RIGHT	UP
s_3	$[2, 1]$	$[2, 1]$	$[1, 2]$	$[0, 3]$	$[2, 1]$	$[1, 2]$
V	$V([2, 1]) = -2$	$V([2, 1]) = -3.6$	$V([1, 2]) = -2.28$	$V([0, 3]) = -2.08$	$V([2, 1]) = -3.54$	$V([1, 2]) = -3.06$
a_3	RIGHT	RIGHT	RIGHT	UP	RIGHT	UP
s_4	$[2, 2]$	$[2, 2]$	$[1, 3]$	$[1, 3]$	$[2, 2]$	$[2, 2]$
V	$V([2, 2]) = -2$	$V([2, 2]) = -1.52$	$V([1, 3]) = -1$	$V([1, 3]) = -1$	$V([2, 2]) = -1.66$	$V([2, 2]) = -1.74$
a_4	RIGHT	RIGHT			RIGHT	RIGHT
s_5	$[2, 3]$	$[2, 3]$			$[2, 3]$	$[2, 3]$
V	$V([2, 3]) = 1$	$V([2, 3]) = 1$			$V([2, 3]) = 1$	$V([2, 3]) = 1$

- Trajectory 1
  - $V([2, 3]) = 1$
    - We can determine this value because $R(s_t, a_t) = +1$ for being in the green state and once we get into the state, the world ends (we cannot transition into any further states), meaning that for all a_t, $(\gamma \sum_{s' \in S} T(s, a, s')V(s')) = 0$.
  - $V(\text{all other states}) = -2$
    - We can determine this value because $R(s_t, a_t) = -2$ and all states initially start out with a value of 0 for all states until after we go through the first iteration.
- Trajectory 2
  - $V([0, 0]) = -2 + \gamma(0.8 \cdot 0 + 0.1 \cdot -2 + 0.1 \cdot -2) = -2 + \gamma(-0.4) = -2 + 0.8 \cdot -0.4 = -2.32$

RL Exercise: Approximate Q-learning

A self-driving car needs to decide whether to Accelerate (**A**) or Brake (**B**) so as to drive to a location without hitting other cars. It receives a reward of +1 if the car moves and does not hit another car, 0 if it does not move, and −2 if it hits another car. The discount factor $\gamma = 1$.

We want to use Approximate Q-learning to learn a good driving policy for the car. The car has sensors that allow it to observe the distance (**D**) to the nearest object and the current speed (**S**). We decide to create a set of four features: f_{AD}, f_{AS}, f_{BD}, f_{BS}. The first two features are for action A and the second two features are for action B. Q values will be approximated by a linear combination of four features, with four weights (one for each feature): w_{AD}, w_{AS}, w_{BD}, w_{BS}.

Suppose that some learning has already happened such that we have $w_{AD} = 1$, but the other weights are all 0. The learning rate $\alpha = 0.5$. Below is the stream of data the car receives as it drives. Compute the weights after each step.

Observed Data	Weights after seeing data
	$w_{AD} = 1, w_{AS} = w_{BD} = w_{BS} = 0$
Initial Sensors: $D = 0, S = 2$ Action: A Reward: −2 Final Sensors: $D = 1, S = 0$	$s[D = 0, S = 2]$, taken action A, $f_{AD} = 0, f_{AS} = 2, f_{BD} = f_{BS} = 0$ $Q(s, \text{A}) = w_{AD} \cdot f_{AD} + w_{AS} \cdot f_{AS} + w_{BD} \cdot f_{BD} + w_{BS} \cdot f_{BS}$ $= 1 \cdot 0 + 0 \cdot 2 + 0 \cdot 0 + 0 \cdot 0 = 0$ $s'[D = 1, S = 0]$, taken action A, $f_{AD} = 1, f_{AS} = 0, f_{BD} = f_{BS} = 0$ $Q(s', \text{A}) = 1 \cdot 1 + 0 \cdot 0 + 0 \cdot 0 + 0 \cdot 0 = 1$ $s'[D = 1, S = 0]$, taken action B, $f_{AD} = f_{AS} = 0, f_{BD} = 1, f_{BS} = 0$ $Q(s', \text{B}) = 1 \cdot 0 + 0 \cdot 0 + 0 \cdot 1 + 0 \cdot 0 = 0$ $\Delta = (-2 + 1.0 \cdot max(Q(s', \text{A}), Q(s', \text{B})) - 0 = -2 + 1.0(1) - 0 = -1$ $w_{AD} \leftarrow w_{AD} + 0.5 \cdot \Delta \cdot f_{AD}(s, A) = 1 + 0.5 \cdot (-1) \cdot 0 =$ (1) $w_{AS} \leftarrow w_{AS} + 0.5 \cdot \Delta \cdot f_{AS}(s, A) = 0 + 0.5 \cdot (-1) \cdot 2 =$ (-1) $w_{BD} =$ (0), $w_{BS} =$ (0)
Initial Sensors: $D = 1, S = 0$ Action: B Reward: 0 Final Sensors: $D = 1, S = 0$	$s[D = 1, S = 0]$, taken action B, $f_{AD} = f_{AS} = 0, f_{BD} = 1, f_{BS} = 0$ $Q(s, \text{B}) = w_{AD} \cdot f_{AD} + w_{AS} \cdot f_{AS} + w_{BD} \cdot f_{BD} + w_{BS} \cdot f_{BS}$ $= 1 \cdot 0 + (-1) \cdot 0 + 0 \cdot 0 + 0 \cdot 0 = 0$ $s'[D = 1, S = 0]$, taken action A, $f_{AD} = 1, f_{AS} = 0, f_{BD} = f_{BS} = 0$ $Q(s', \text{A}) = 1 \cdot 1 + (-1) \cdot 0 + 0 \cdot 0 + 0 \cdot 0 = 1$ $s'[D = 1, S = 0]$, taken action B, $f_{AD} = f_{AS} = 0, f_{BD} = 1, f_{BS} = 0$ $Q(s', \text{B}) = 1 \cdot 0 + (-1) \cdot 0 + 0 \cdot 1 + 0 \cdot 0 = 0$ $\Delta = (0 + 1.0 \cdot max(Q(s', \text{A}), Q(s', \text{B})) - 0 = 0 + 1.0(1) - 0 = 1$ $w_{AD} =$ (1), $w_{AS} =$ (-1) $w_{BD} \leftarrow w_{BD} + 0.5 \cdot \Delta \cdot f_{BD}(s, B) = 0 + 0.5 \cdot 1 \cdot 1 =$ (0.5) $w_{BS} \leftarrow w_{BS} + 0.5 \cdot \Delta \cdot f_{BS}(s, B) = 0 + 0.5 \cdot 1 \cdot 0 =$ (0)

Given the learned weights, suppose that the sensors read $D = 1, S = 1$. Which action would be preferred?

- At state $s[D = 1, S = 1]$, if take action A, $Q(s, \text{A}) = 1 \cdot 1 + (-1) \cdot 1 + 0.5 \cdot 0 + 0 \cdot 0 = 0$
- At state $s[D = 1, S = 1]$, if take action B, $Q(s, \text{B}) = 1 \cdot 0 + (-1) \cdot 0 + 0.5 \cdot 1 + 0 \cdot 1 =$ (0.5)

Hence, action B will be preferred as the corresponding Q-value (i.e., $Q(s, \text{B})$) is greater.

Conditional Probability Table	Number of Parameters
$p(D_1)$	1
$p(D_2\|D_1)$	2
$p(D_3\|D_1, D_2)$	4
$p(S_1\|D_1, D_2, D_3)$	8
$p(S_2\|D_1, D_2, D_3, S_1)$	16
$p(S_3\|D_1, D_2, D_3, S_1, S_2)$	32
$p(S_4\|D_1, D_2, D_3, S_1, S_2, S_3)$	64
Total Number of Parameters	127

(We can see there is no saving relative to specifying the joint probability distribution directly, which would require $2^7 - 1 = 127$ numbers.)

- **Q:** What diseases do we gain information about when observing the fourth symptom ($S_4 = true$)?

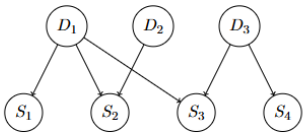
  A: We have independence relations $I(D_1, S_4)$ (since the path is blocked without observing S_3 and $I(D_2, S_4)$ (since the path is blocked at both S_2 and S_3). What is left is dependence between D_3 and S_4. Thus, we only learn information about D_3.

- **Q:** Suppose we know that the third symptom is present ($S_3 = true$). What does observing the fourth symptom ($S_4 = true$) tell us now?

  A: With $S_3 = true$, observing $S_4 = true$ now also gives us information about D_1 (via 'explaining away', or using d-separation, because the D_1 to S_4 path is no longer blocked at S_3). We still don't learn any information about D_2 because the D_2 to S_4 path remains blocked at S_2.

- **Q:** Draw a Bayesian network for this problem with the variable ordering $D_1, D_2, D_3, S_1, S_2, S_3, S_4$.

  A: Note that there are many valid networks (depending on the chosen variable ordering), some more efficient (i.e. requiring fewer parameters) than others. Here is a compact representation that comes from variable ordering $D_1, D_2, D_3, S_1, S_2, S_3, S_4$. (Recall that all dependencies to earlier variables need to be indicated with edges).



- **Q:** Write down the expression for the joint probability distribution given this network.

  A: $p(D_1, D_2, D_3, S_1, S_2, S_3, S_4)$

  $= p(D_1)p(D_2)p(D_3)p(S_1|D_1)p(S_2|D_1, D_2)p(S_3|D_1, D_3)p(S_4|D_3)$

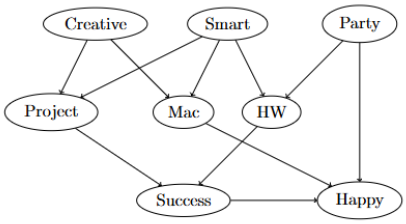
- **Q:** How many parameters are required to describe this joint distribution?

  A:

Conditional Probability Table	Number of Parameters
$p(D_1)$	1
$p(D_2)$	1
$p(D_3)$	1
$p(S_1\|D_1)$	2
$p(S_2\|D_1, D_2)$	4
$p(S_3\|D_1, D_3)$	4
$p(S_4\|D_3)$	2
Total Number of Parameters	15

- **Q:** How many parameters would be required to represent the CPTs in a Bayesian network if there were no conditional independences between variables?

  A: The network would be structured as a clique, and considering order $D_1, D_2, D_3, S_1, S_2, S_3, S_4$, the number of parameters for the CPTs would be $1 + 2 + 4 + 8 + 16 + 32 + 64 = 127$.



- **Q:** True or False: *Party* is independent of *Success* given *HW*.

  A: False; there is a path that is not blocked: *Party* − *HW* − *Smart* − *Project* − *Success* has neither a converging arrows not in the set of evidence or a non-converging arrows in the set.

- **Q:** True or False: *Creative* is independent of *Happy* given *Mac*.

  A: False; there is a path that is not blocked: *Creative* − *Project* − *Success* − *Happy*

- **Q:** True or False: *Party* is independent of *Smart* given *Success*.

  A: False; there is a path that is not blocked between *Party* and *Smart*: the path *Party* − *HW* − *Success* is not blocked because the converging arrows node at *HW* has a descendant (*Success*) in the evidence.

- **Q:** True or False: *Party* is independent of *Creative* given *Happy*.

5.1 Exercise: Variable Elimination

Consider the Bayesian network described in above, and assume the following Conditional Probability Table (CPT). Let $x_i \in \{0, 1\}$ denote the values that variable X_i can take. Our goal is to find $p(x_4)$.

x_1	$p(x_1)$
0	0.3
1	0.7

x_2	$p(x_2)$
0	0.6
1	0.4

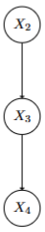
x_3	x_1	x_2	$p(x_3\|x_1, x_2)$
0	0	0	0.5
0	0	1	0.2
0	1	0	0.9
0	1	1	0.5
1	0	0	0.5
1	0	1	0.8
1	1	0	0.1
1	1	1	0.5

x_4	x_3	$p(x_4\|x_3)$
0	0	0.7
0	1	0.1
1	0	0.3
1	1	0.9

1. Eliminate X_1 first. Draw the resulting Bayesian network and compute the CPT.
2. Eliminate X_3 first. Draw the resulting Bayesian network and compute the CPT.
3. How many sum-product calculations do each of these variable elimination orders require? Which one is preferable?

Solution

1. The resulting network is:



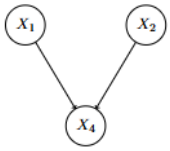
The variable elimination process eliminates X_1 by marginalizing out X_1: $p(x_3|x_2) = \sum_{x_1} p(x_3|x_1, x_2)p(x_1)$. For example:

$$\begin{aligned} p(X_3 = 0|X_2 = 0) &= \sum_{x_1 \in \{0,1\}} p(X_3 = 0|X_1 = x_1, X_2 = 0)p(X_1 = x_1) \\ &= 0.5 \cdot 0.3 + 0.9 \cdot 0.7 \\ &= 0.78 \end{aligned}$$

This is a sum-product calculation, and we need to do one for each value of X_2 and X_3. Thus, there are four sum-product calculations in total. The resulting CPT is:

x_3	x_2	$p(x_3\|x_2)$
0	0	0.78
0	1	0.41
1	0	0.22
1	1	0.59

2. The resulting network is



The variable elimination process eliminates X_3 by marginalizing out X_3: $p(x_4|x_1, x_2) = \sum_{x_3} p(x_4|x_3)p(x_3 \mid x_1, x_2)$. This would be the first intermediate term. For example:

$$\begin{aligned} p(X_4 = 0|X_1 = 0, X_2 = 0) &= \sum_{x_3 \in \{0,1\}} p(X_4 = 0|X_3 = x_3)p(X_3 = x_3|X_1 = 0, X_2 = 0) \\ &= 0.7 \cdot 0.5 + 0.1 \cdot 0.5 \\ &= 0.40 \end{aligned}$$

We need to do this for each combination of values for X_1, X_2 and X_4. Thus, there are eight sum-product calculations in total. The resulting CPT is:

x_4	x_1	x_2	$p(x_4\|x_1, x_2)$
0	0	0	0.40
0	0	1	0.22
0	1	0	0.64
0	1	1	0.40
1	0	0	0.60
1	0	1	0.78
1	1	0	0.36
1	1	1	0.60

3. In these variable elimination operations, we need to compute intermediate terms. The cost of computing these depends on the number of variables that they mention, since each variable increases the number of required sum-product calculations by a factor of $k = 2$.

   For the first ordering, the intermediate terms are:

   - $p(x_3 \mid x_2)$: mentions x_2 and x_3, and thus requires four sum-product calculations (for each row in the original CPT)
   - $p(x_3)$: mentions x_3 and thus requires two sum-product calculations
   - $p(x_4)$: mentions x_4 and thus requires two sum-product calculations

We have a total of $4 + 2 + 2 = 8$ sum-product calculations.

For the second ordering, the intermediate terms are:

- $p(x_4 \mid x_1, x_2)$: mentions x_1, x_2 and x_4, and thus requires eight sum-product calculations (for each row in the original CPT)
- $p(x_4 \mid x_1)$: mentions x_1 and x_4, and thus requires four sum-product calculations
- $p(x_4)$: mentions x_4 and thus requires two sum-product calculations

We have a total of $8 + 4 + 2 = 14$ sum-product calculations.

Thus, we see that the first ordering is preferable since it requires fewer computational steps.

Mid-term

(b) **(2 pts) Admissible Heuristics.** If $f(s)$, $g(s)$, and $h(s)$ are all admissible heuristics then which of the following are also guaranteed to be admissible heuristics:

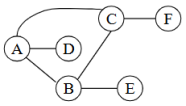
- ○ $f(s) + g(s) + h(s)$
- ● $f(s)/3 + g(s)/3 + h(s)/3$
- ● $f(s)/6 + g(s)/3 + h(s)/2$
- ○ $f(s) * g(s) * h(s)$
- ● $min(f(s), g(s), h(s))$
- ● $max(f(s), g(s), h(s))$

Hint: Suppose the actual cost is $v(s)$, then we know that "$f(s)$, $g(s)$, and $h(s)$ being all admissible heuristics" means that $f(s), g(s), h(s) \leq v(s)$.
If $h(s)$ is the maximum among $\{f(s), g(s), h(s)\}$, we can derive the following:

1. $f(s)/3 + g(s)/3 + h(s)/3 \leq h(s)/3 + h(s)/3 + h(s)/3 = h(s) \leq v(s)$. So we have: $f(s)/3 + g(s)/3 + h(s)/3 \leq v(s)$, which means that $f(s)/3 + g(s)/3 + h(s)/3$ is admissible.
2. $f(s)/6 + g(s)/3 + h(s)/2 \leq h(s)/6 + h(s)/3 + h(s)/2 = h(s) \leq v(s)$. So we have: $f(s)/6 + g(s)/3 + h(s)/2 \leq v(s)$, which means that $f(s)/6 + g(s)/3 + h(s)/2$ is admissible.

If you repeat the above derivation with different assumptions of the maximum among $\{f(s), g(s), h(s)\}$, you will arrive at the same conclusion, which means that it does not matter which is the maximum among $\{f(s), g(s), h(s)\}$, both $f(s)/3 + g(s)/3 + h(s)/3$ and $f(s)/6 + g(s)/3 + h(s)/2$ are always admissible.

(c) **(1 pt each - total of 2 pts) CSP.** The graph below is a constraint graph for a CSP that has only binary constraints. Initially, no variables have been assigned. For each of the following scenarios, mark all variables for which the specified filtering might result in their domain being changed.



i) A value is assigned to A. Which domains might be changed as a result of running forward checking for A?
○ A ● B ● C ● D ○ E ○ F

ii) A value is assigned to A, and then forward checking is run for A. Then a value is assigned to B. Which domains might be changed as a result of running forward checking for B?
○ A ○ B ● C ○ D ● E ○ F

(c) **(1 pt each - total of 5 pts) True or False** Circle the correct answer.

i) (T) F The game, Tic-tac-toe, is fully-observable.

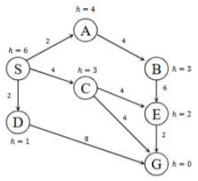
ii) (T) F In a graph where the goal is neither the root nor at depth one, iterative deepening search will definitely expand more nodes than breadth-first search.

iii) (T) F A* search with the heuristic h(n) = 0 is equivalent to uniform-cost search.

iv) T (F) A two variable integer programming problem cannot be solved by the graphical method.

v) (T) F The union of two convex sets may or may not be convex.

goal check when nodes are visited, not when their parent is expanded to create them as children. *If a state is visited more than once, write it each time.*



(a) **(3 pts)** Iterative deepening depth first search:

Visited order: S, S, A, C, D, S, A, B, C, E, G

Solution (path length: 8): S, C, G

(b) **(3 pts)** Uniform Cost Search:

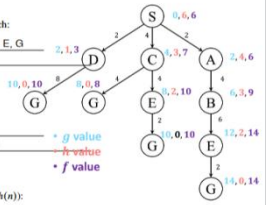
Visited order: S, A, D, C, B, E, G

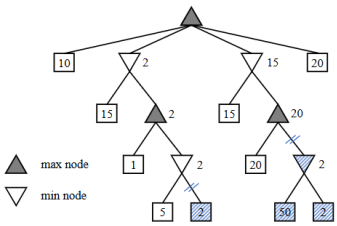
Solution (path length: 8): S, C, G

(a) **(3 pts)** A* Search (assume $f(n) = g(n) + h(n)$):

Visited order: S, D, A, C, G

Solution (path length: 8): S, C, G





(a) **(2 pts)** Fill in the mini-max values for each of the nodes in the tree that aren't leaf nodes. What is the minimax value for the root?

20

(b) **(5 pts)** If α-β pruning were run on this tree, which branches would be cut? Mark the branches with a slash or a swirl (like a cut) and shade the nodes that don't get explored.

See pic above.

(c) **(2 pts)** Is there another ordering for the children of the root for which more pruning would result? If so, state the order.

Yes, if we had the children ordered as 20, 15, 10, 2.

The classes are:

1. Class 1 - Circuits and Systems: meets from 8:00 - 9:00 am
2. Class 2 - Digital Logic Fundamentals: meets from 8:30 - 9:30 am
3. Class 3 - Electromagnetic Fields and Waves: meets from 9:00 - 10:00 am
4. Class 4 - Control Systems: meets from 9:00 - 10:00 am
5. Class 5 - Microprocessors and Digital Systems: meets from 9:30 - 10:30 am

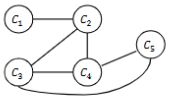
The professors are:

1. Professor A, who is qualified to teach Classes 3 and 4.
2. Professor B, who is qualified to teach Classes 2, 3, 4, and 5.
3. Professor C, who is qualified to teach Classes 1, 2, 3, 4, and 5.

(a) **(2 pts)** Formulate this problem as a CSP problem in which there is one variable per class, stating the domains, and constraints. Constraints should be specified formally and precisely, but may be implicit rather than explicit.

Variables	Domains	Constraints
C_1	{C}	$C_1 \neq C_2$
C_2	{B, C}	$C_2 \neq C_3$
C_3	{A, B, C}	$C_2 \neq C_4$
C_4	{A, B, C}	$C_3 \neq C_4$
C_5	{B, C}	$C_3 \neq C_5$
		$C_4 \neq C_5$

(b) **(2 pts)** Draw the constraint graph associated with your CSP.



(c) **(1 pts)** Give one solution to this CSP (i.e., a solution that satisfies all constraints).

$C_1 = C, C_2 = B, C_3 = C, C_4 = A, C_5 = B.$

One other solution: $C_1 = C, C_2 = B, C_3 = A, C_4 = C, C_5 = B.$

5 Linear Programming

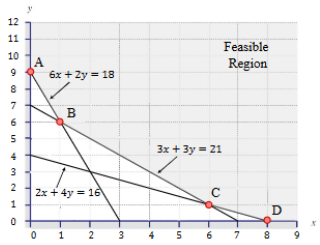
In order to supplement his daily diet, someone wishes to take some Xtravit and some Yeastalife tablets. Their content of iron, calcium, and vitamins (in milligrams per tablet) are shown in the table below.

Tablet	Iron	Calcium	Vitamin
Xtravit	6	3	2
Yeastalife	2	3	4

(a) **(2 pts)** By taking x tablets of Xtravit and y tablets of Yeastalife, the person expects to receive at least 18 milligrams of iron, 21 milligrams of calcium, and 16 milligrams of vitamins. Write these conditions down as three inequalities in terms of x and y.

$$6x + 2y \geq 18$$
$$3x + 3y \geq 21$$
$$2x + 4y \geq 16$$

(b) **(3 pts)** In the provided coordinate plane below, illustrate the region of those points (x, y) which simultaneously satisfy all the constraints.



(c) **(2 pts)** If Xtravit tablets cost \$10 each and the Yeastalife tablets cost \$5 each, how many tablets of each should the person take in order to satisfy the above requirements at minimum cost?

Minimum cost occurs at point B ($x = 1$, $y = 6$), and the cost is given by

$$P = 10x + 5y = 10 \times 1 + 5 \times 6 = 10 + 30 = 40$$