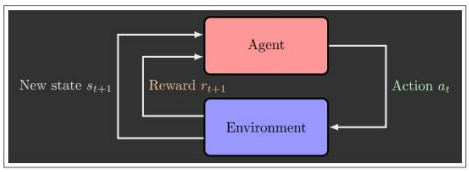
02 search problem

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

- States: (x,y) position
- Actions: NEWS
- Successor: update location
- Goal Test: Is (x,y) == END

Problem: Eat-All-Dots

- States: (x,y), boolean for each dot
- Actions: NEWS
- Successor: update location, boolean for dots
- Goal Test: All dot booleans are false

03 uninformed-search

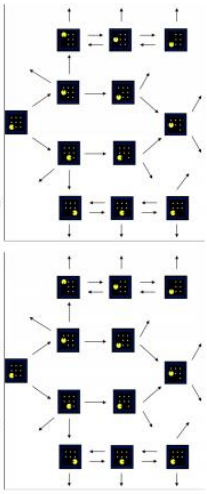
STATE SPACE GRAPHS

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?

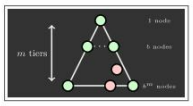
DEPTH-FIRST SEARCH (DFS) PROPERTIES

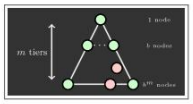
Strategy: expand the deepest node first

Implementation: fringe is a FILO or LIFO stack

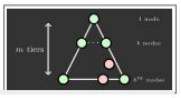
What nodes DFS expand?

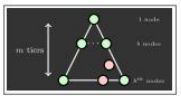
- Some left prefix of the tree.
- Could process the whole tree!
- If m is finite, takes time $\mathcal{O}(b^m)$





BREADTH-FIRST SEARCH





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.

BREADTH-FIRST SEARCH (BFS) PROPERTIES

Strategy: expand the shallowest node first

Implementation: fringe is a FIFO queue

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)

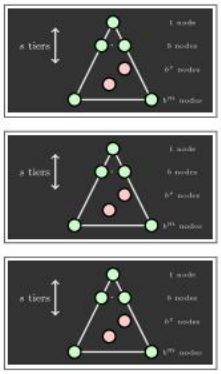
UCS PROPERTIES

What nodes UCS expand?

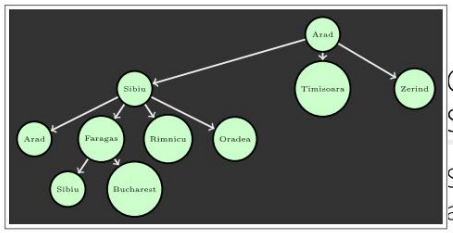
- Processes all nodes with cost less than cheapest solution!
- If that solution costs C^* and arcs cost at least ϵ, then the "effective depth" is roughly $\left(\frac{C^*}{\epsilon}\right)$
- Takes time $\mathcal{O}\left(b^{\frac{C^*}{\epsilon}}\right)$ (exponential in effective depth)

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

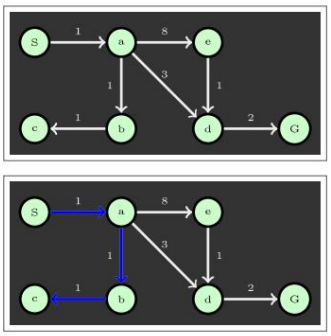
Worst-case: like a badly-guided DFS

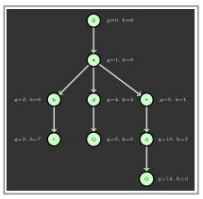
COMBINING UCS AND GREEDY

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





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

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

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

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

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}

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.

WHAT IS SEARCH FOR?

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

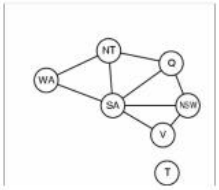
- Planning: sequences of actions
- The path to the goal is the important thing
- Paths have various costs, depths
- Heuristics give problem-specific guidance
- Identification: assignments to variables
- The goal itself is important, not the path
- All paths at the same depth (for some formulations)
-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

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.



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)

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

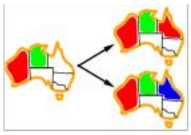
ORDERING: MINIMUM REMAINING VALUES

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

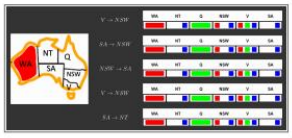


- Given a choice of variable, choose the least constraining value
- i.e., the one that rules out the fewest values in the remaining variables

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

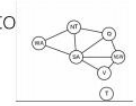
```
function AC-3(csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X_1, X_2, ..., X_n}
  local variables: queue, a queue of arcs, initially all the arcs in csp
  while queue is not empty do
    (X_i, X_j) ← REMOVE-FIRST(queue)
    if REMOVE-INCONSISTENT-VALUES(X_i, X_j) then
      for each X_k in NEIGHBORS[X_i] do
        add (X_k, X_i) to queue

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

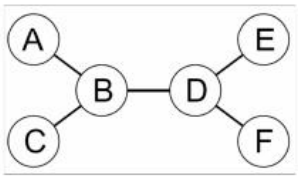
- Tasmania and mainland are independent subproblems

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

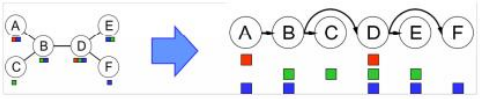
Compare to general CSPs, where worst-case time is $\mathcal{O}(d^n)$

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:

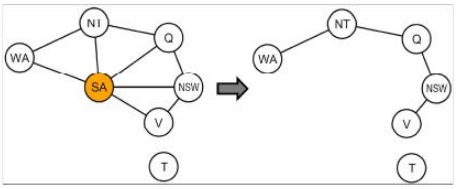
- Order: Choose a root variable, order variables so that parents precede children



- 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** $S \times A \rightarrow S$
- **Terminal Test** $S \rightarrow \{t, f\}$
- **Terminal Utilities** $S \times P \rightarrow R$

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

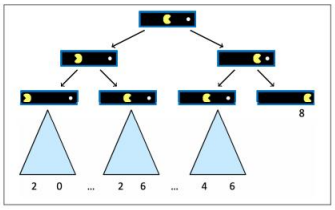
GAME THEORY

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: $V(s) = \max_{s' \in children(s)} V(s')$

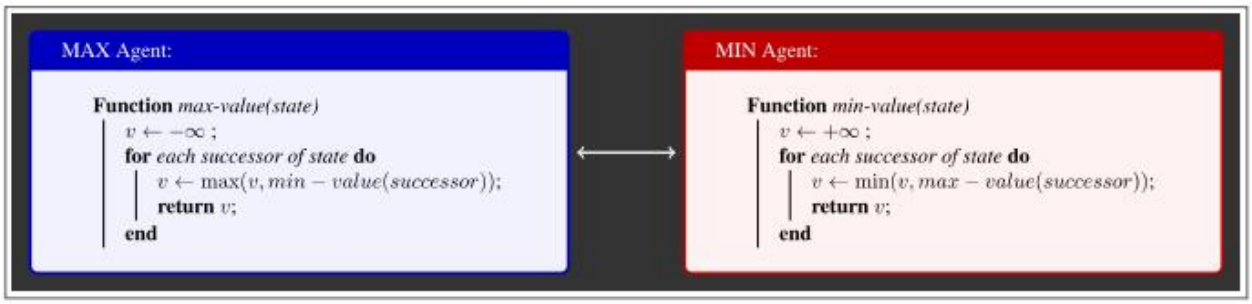
States Under Agent's Control: $V(s) = \max_{s' \in successors(s)} V(s')$

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

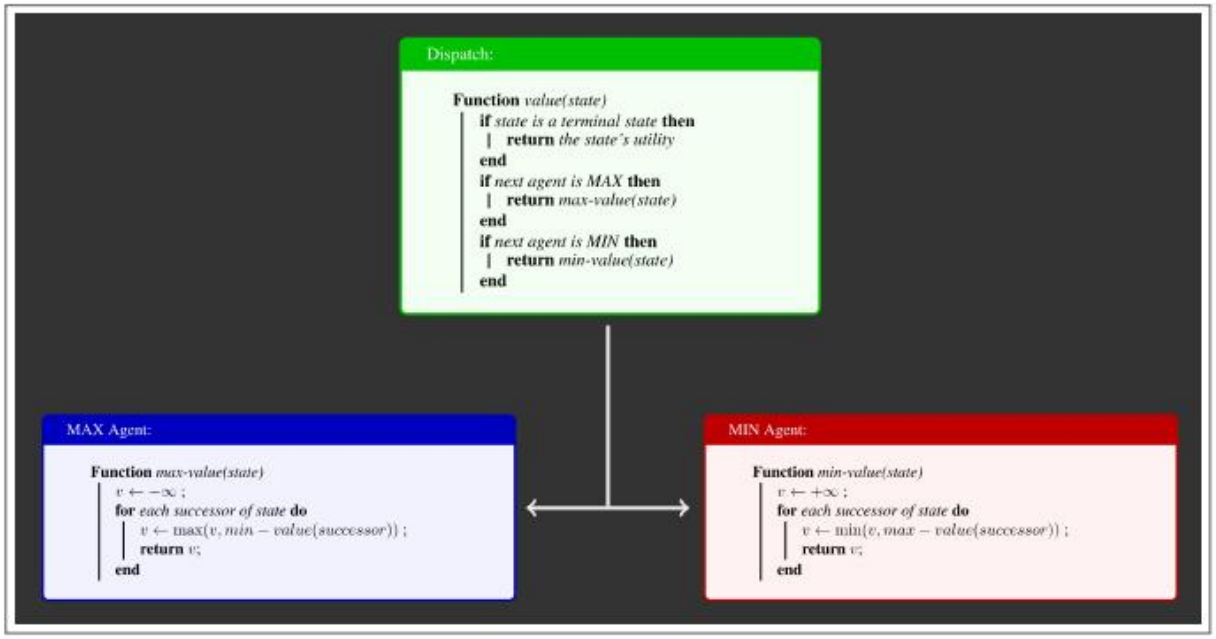
Terminal States: $V(s)$ known

MINIMAX VALUES

MINIMAX IMPLEMENTATION



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, $b \approx 35$, $m \approx 100$

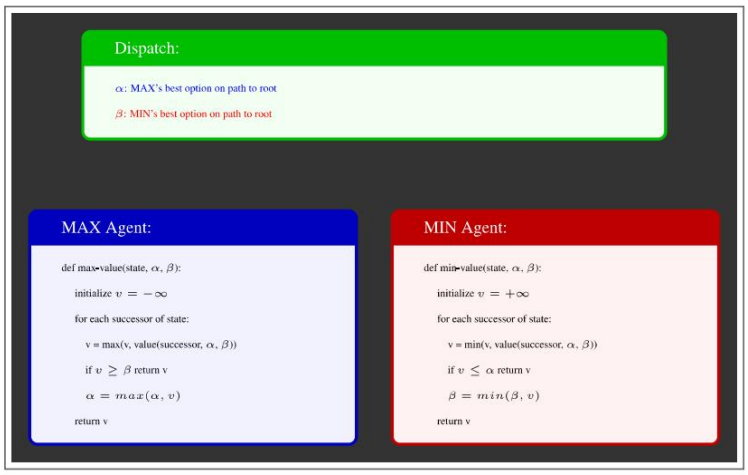
$A - B$ PRUNING

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

$A - B$ 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

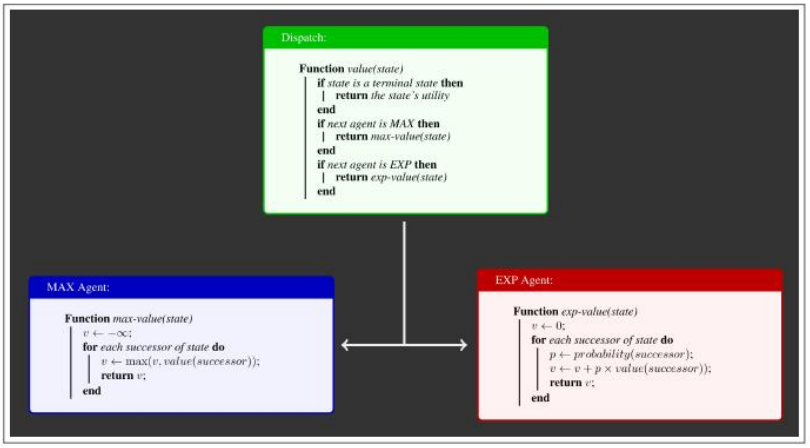
With "perfect ordering":

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

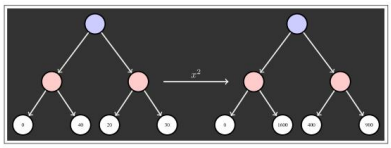
This is a simple example of metareasoning (computing about what to compute)

08

EXPECTIMAX IMPLEMENTATION



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

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

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

OPTIMIZATION FORMULATION

Diet Problem

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

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

10

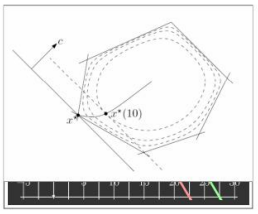
SHAPES IN HIGHER DIMENSIONS

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**

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 y_{IP}^* be the optimal objective of an integer program P.

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

Let y_{LP}^* be the optimal objective of the LP-relaxed version of P.

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

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

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

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

11 optimization

OPTIMIZATION PROBLEM: DEFINITION

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

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

- $f(\mathbf{x}) = 1$: Feasibility problem
- Simple functions
- Linear function $f(\mathbf{x}) = \mathbf{a}^T\mathbf{x}$
- 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

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)

OPTIMIZATION PROBLEM: EXAMPLE

Example: Traveling Salesman Problem (TSP)

- Problem: n cities, distance from city i to city j is $d(i, j)$, 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
- x_i is the index of the i-th city being visited
- Feasible set $\mathcal{F} = \{\mathbf{x} :$ "each city visited exactly once"$\}$

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

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

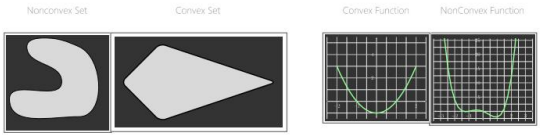
12 convex optimization

Convex Optimization Problem:

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

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 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 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 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 $\mathcal{F}$.

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

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

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

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., $P_{\mathcal{F}} = \underset{\mathbf{x}' \in \mathcal{F}}{\operatorname{argmin}} \|\mathbf{x} - \mathbf{x}'\|_2^2$

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: 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., $\|\mathbf{x}^{i+1} - \mathbf{x}^i\| \leq \epsilon$

CityU CS5491 AI Midterm Exam Solution

(Spring 2024)

Name: Instructor Student ID: Solution

Instructions

- Place your name and your student ID number on the front page.
- The maximum possible score on this exam is 40. You have approximately 1 hour and 50 minutes.
- This is an open-book exam, you may use any notes or books that are in physical printout format. No electronic devices are allowed.
- Please answer clearly and succinctly. If an explanation is requested, think carefully before writing. Points may be removed for rambling answers. If a question is unclear or ambiguous, feel free to make the additional assumptions necessary to produce the answer. State these assumptions clearly; you will be graded on the basis of the assumption as well as subsequent reasoning.
- Good luck!

Problem	Points
Problem 1	/10
Problem 2	/9
Problem 3	/9
Problem 4	/5
Problem 5	/7
Total	/40

1 Multiple Choice Questions

Note: *incorrect answers will incur negative points* proportional to the number of choices. For example, a 1 point true-false question will receive 1 point if correct, -1 if incorrect, and zero if left blank. Only make informed guesses.

(a) **(1 pt) Who are you?** Write your name and student ID at the top of the cover page.

(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:

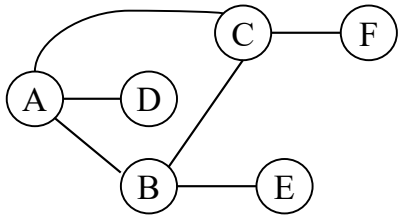
- ○ $f(s) + g(s) + h(s)$
- ● $f(s)/3 + g(s)/3 + h(s)/3$
- ● $f(s)/6 + g(s)/3 + h(s)/2$
- ○ $f(s) * g(s) * h(s)$
- ● $min(f(s), g(s), h(s))$
- ● $max(f(s), g(s), h(s))$

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.

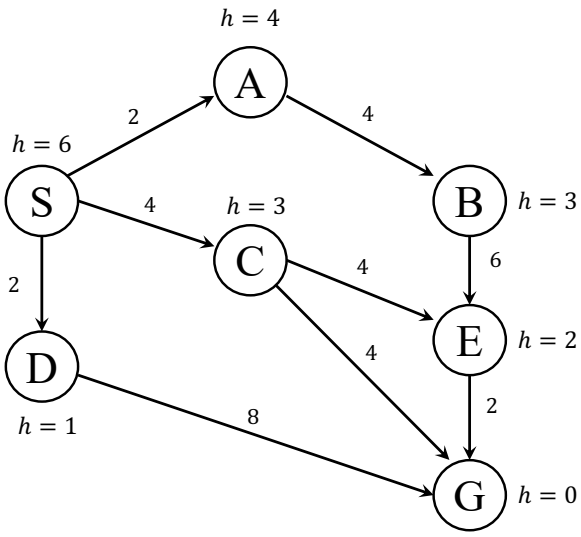
iii) (T) F A* search with the heuristic h(n) = 0 is equivalent to uniform-cost search.

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.

2 Informed Search

Given the graph below, suppose you want to go from start state "S" to goal state "G", write down the order in which the states are *visited* and the path found by the following search algorithms. Ties (e.g., which child to first explore in depth-first search) should be resolved alphabetically (i.e., prefer A before Z). Remember to include the start and goal states in your answer. Assume that algorithms execute the 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:

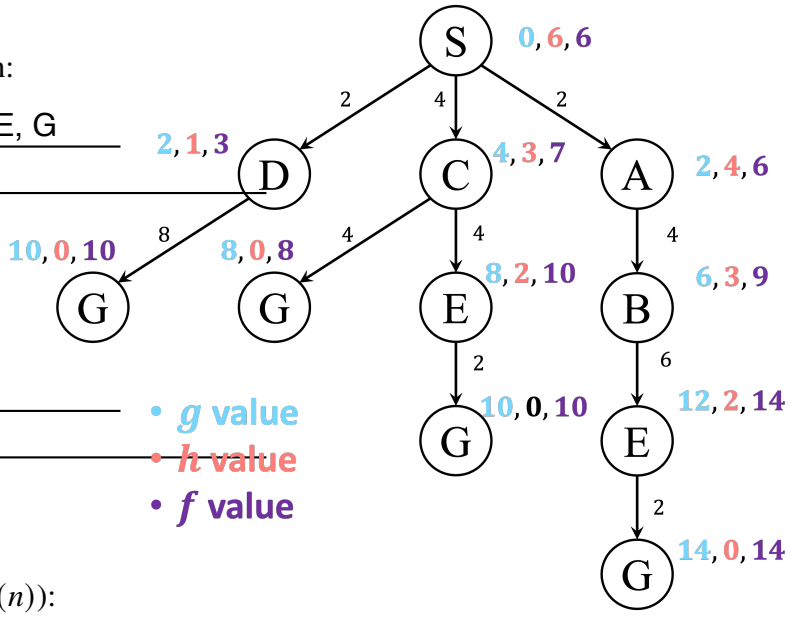
Visited order: S, S, A, C, D, S, A, B, C, E, G

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



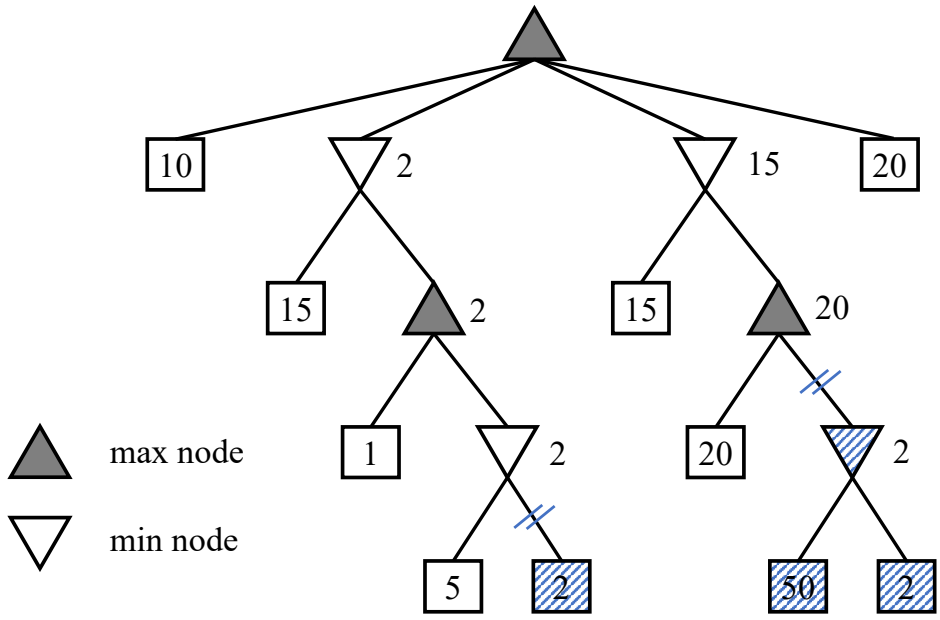
(a) (**3 pts**) A* Search (assume $f(n) = g(n) + h(n)$):

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

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

3 Minimax Search

Consider the mini-max tree, whose root is a max node, shown below. Assume that children are explored left to right.



(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.

4 Course Scheduling

You are in charge of scheduling electrical engineering classes that meet on Mondays, Wednesdays, and Fridays. There are 5 classes that meet on these days and 3 professors who will be teaching these classes. You are constrained by the fact that each professor can only teach one class at a time.

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.

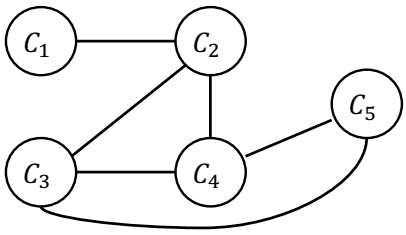
(a) (**2 pts**) Formulate this problem as a CSP problem in which there is one variable per class, stating the domains, and constraints. Constraints should be specified formally and precisely, but may be implicit rather than explicit.

Variables	Domains
C_1	{C}
C_2	{B, C}
C_3	{A, B, C}
C_4	{A, B, C}
C_5	{B, C}

Constraints

$C_1 \neq C_2$
$C_2 \neq C_3$
$C_2 \neq C_4$
$C_3 \neq C_4$
$C_3 \neq C_5$
$C_4 \neq C_5$

(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

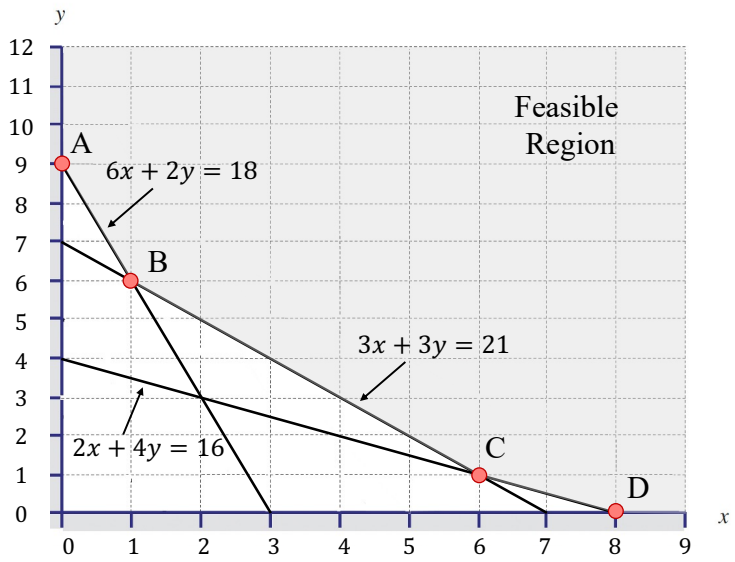
In order to supplement his daily diet, someone wishes to take some Xtravit and some Yeastalife tablets. Their content of iron, calcium, and vitamins (in milligrams per tablet) are shown in the table below.

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

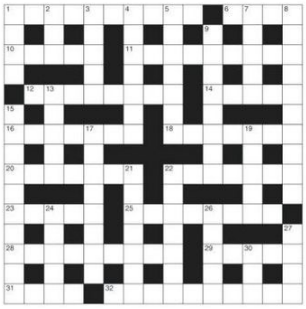
$$P = 10x + 5y = 10 \times 1 + 5 \times 6 = 10 + 30 = 40$$

1 Multiple Choice Questions

Note: *incorrect answers will incur negative points* proportional to the number of choices. For example, a 1 point true-false question will receive 1 point if correct, -1 if incorrect, and zero if left blank. Only make informed guesses.

(a) (**1 pt**) **Who are you?** Write your name and student ID at the top of the cover page.

(b) (**1 pt each - total of 4 pts**) **Types of Agents.** You are developing an agent that solves crossword puzzles (like the one pictured to the right) using an exhaustive dictionary of possible words. States are partially completed puzzles and actions place a word on the puzzle. On each line below, we've listed two possible environmental aspects; circle the one that better describes the crossword puzzle environment.



i) [fully observable] vs. partially observable

ii) [single agent] vs. multi-agent

iii) stochastic vs. [deterministic]

iv) [discrete] vs. continuous

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

i) [T] F A* search with a heuristic that is not completely admissible may still find the shortest path to the goal state.

ii) T [F] Doubling your computer's speed allows you to double the depth of a tree search given the same amount of time.

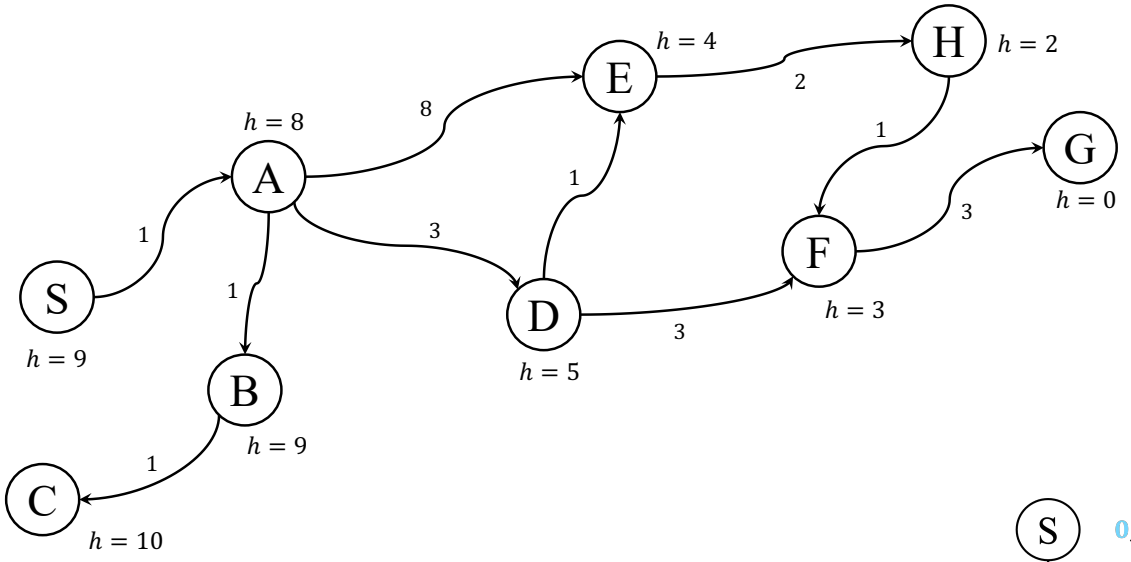
iii) [T] F Backtracking search on CSPs, while generally much faster than general purpose search algorithms like A*, still requires exponential time in the worst case.

iv) [T] F An agent that uses Minimax search, which assumes an adversary behaves optimally, may well achieve a better score when playing against a suboptimal adversary than the agent would against an optimal adversary.

v) T [F] For solving an integer programming problem, it is sufficient to consider the integer points around the corresponding LP solution.

2 Informed Search

Given the graph below, suppose you want to go from start state "S" to goal state "G", write down the order in which the states are *visited* and the path found by the following search algorithms. Ties (e.g., which child to first explore in depth-first search) should be resolved alphabetically (i.e., prefer A before Z). Remember to include the start and goal states in your answer. Assume that algorithms execute the goal check when nodes are visited, not when their parent is expanded to create them as children. Do not expand any node more than once (graph search implementation).



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

Visited order: S A B C D E F H G

Solution (path length: 10): S A D F G

(b) (**3 pts**) Greedy Search:

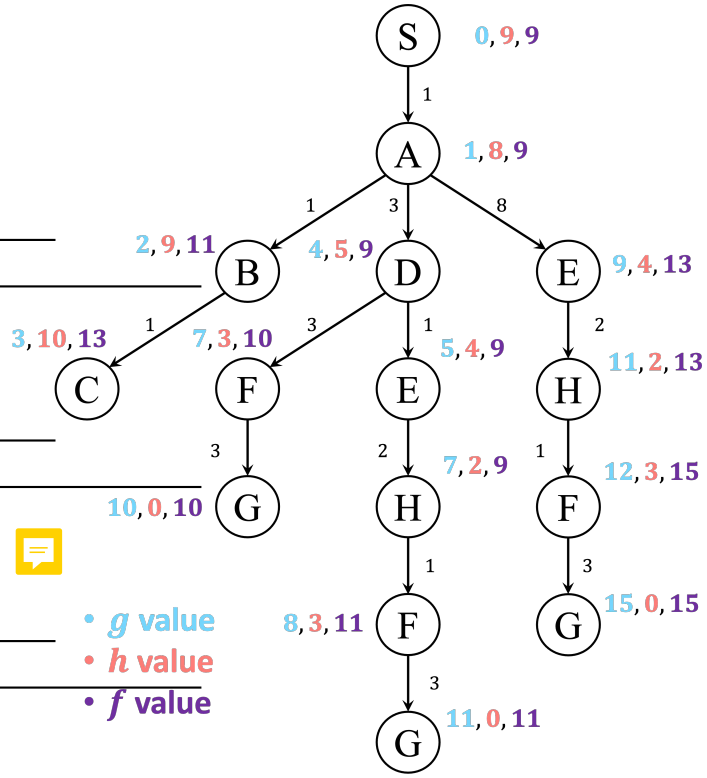
Visited order: S A E H F G

Solution (path length: 15): S A E H F G

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

Visited order: S A D E H F G

Solution (path length: 10): S A D F G



3 Course Scheduling

You are in charge of scheduling computer science classes that meet on Mondays, Wednesdays, and Fridays. There are 5 classes that meet on these days and 3 professors who will be teaching these classes. You are constrained by the fact that each professor can only teach one class at a time.

The classes are:

1. Class 1 - Intro to Programming: meets from 8:00 - 9:00 am
2. Class 2 - Intro to Artificial Intelligence: meets from 8:30 - 9:30 am
3. Class 3 - Natural Language Processing: meets from 9:00 - 10:00 am
4. Class 4 - Computer Vision: meets from 9:00 - 10:00 am
5. Class 5 - Machine Learning: meets from 10:30 - 11:30 am

The professors are:

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

(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
C_1	{A, C}
C_2	{A}
C_3	{B, C}
C_4	{B, C}
C_5	{A, B}

Constraints

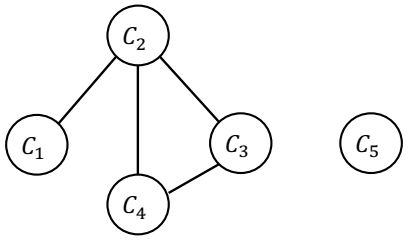
$C_1 \neq C_2$

$C_2 \neq C_3$

$C_2 \neq C_4$

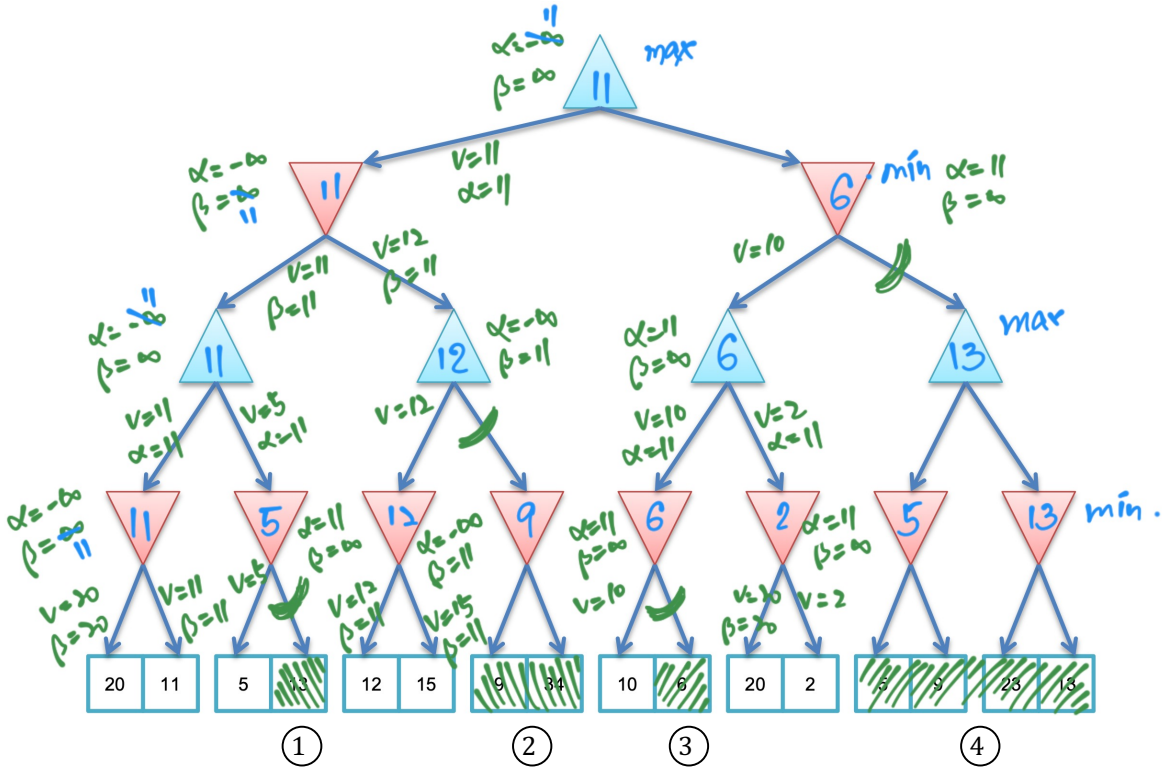
$C_3 \neq C_4$

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



4 Adversarial Search

Consider the mini-max tree, whose root is a max node, shown below. Assume that children are explored left to right.



(a) (**3 pts**) Fill in the mini-max values for each of the nodes in the tree that aren't leaf nodes

(b) (**6 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 leaf nodes that don't get explored.

initialize: $\alpha = -\infty$
$\beta = -\infty$

at max node: prune if $v \geq \beta$
else: update $\alpha = max(\alpha, v)$

at min node: prune if $v \leq \alpha$
else: update $\beta = min(\beta, v)$

prune:

①: because $(v = 5) < (\alpha = 11)$

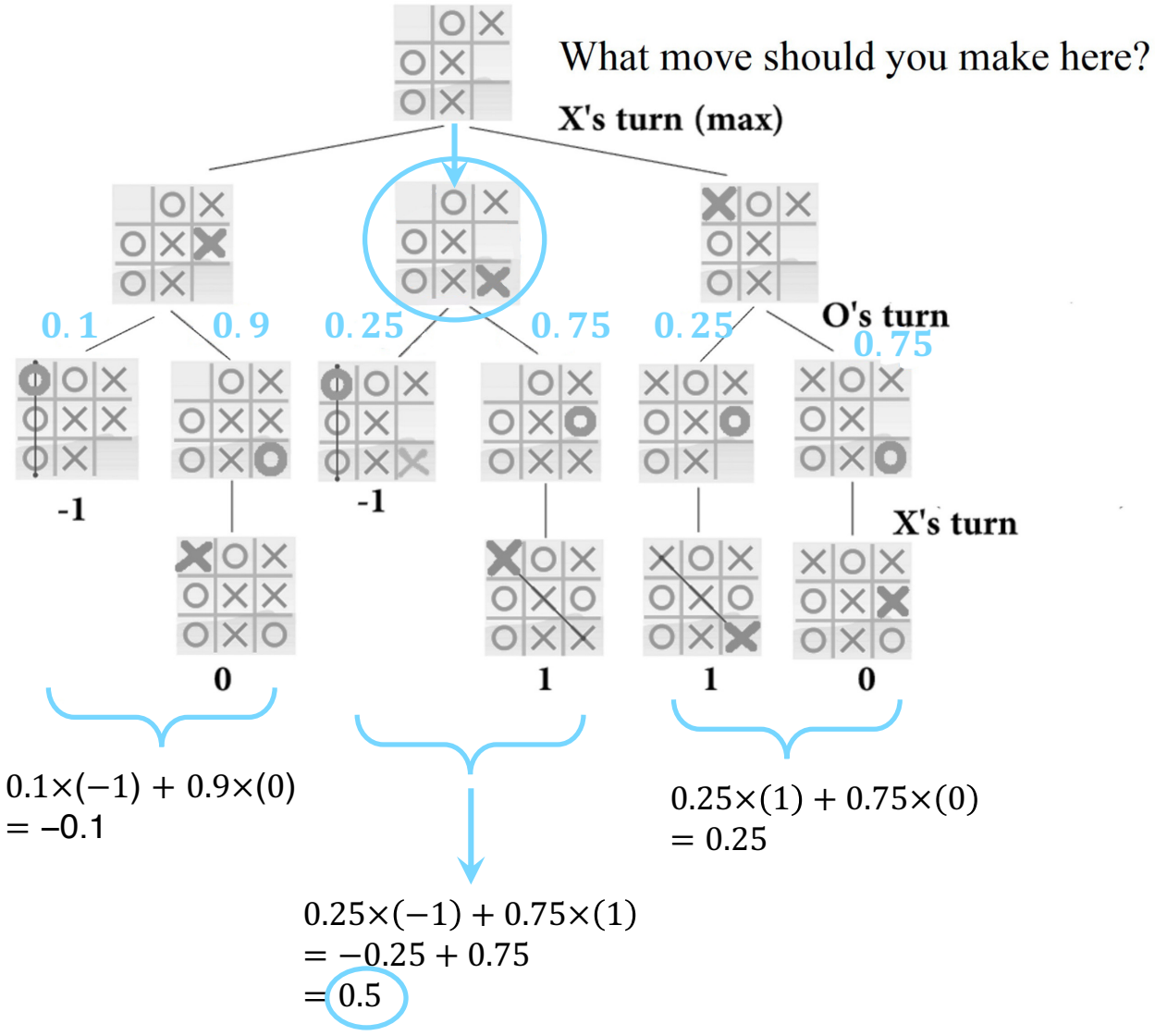
②: because $(v = 12) > (\beta = 11)$

③: because $(v = 10) < (\alpha = 11)$

④: because $(v = 10) < (\alpha = 11)$

5 Expectimax

(**4 pts**) You, are playing tic tac toe as the "X" player. You opponent, Olivia ("O" player) is a child, who is playing randomly. Since Olivia is very short, she is much more likely to fill "O"s into lower rows. Specifically, you know that Olivia is three times as likely to choose a space on the middle row than the top row, and three times again more likely to choose a space on the bottom row than the middle row. What move should you make to maximize your chance of winning? (Circle the board position and provide justifications)



6 Linear Programming

Ann and Margaret run a small business in which they work together making blouses and skirts. Each blouse takes 1 hour of Ann's time together with 1 hour of Margaret's time. Each skirt involves Ann for 1 hour and Margaret for half an hour. Ann has 7 hours available each day and Margaret has 5 hours each day. Suppose they get \$8 profit on a blouse and \$6 on a skirt. Find the number of blouses and skirts that they should make to maximize daily profit. (Note that they could just make blouses or they could just make skirts or they could make some of each. However, a partial blouse or skirt is not allowed).

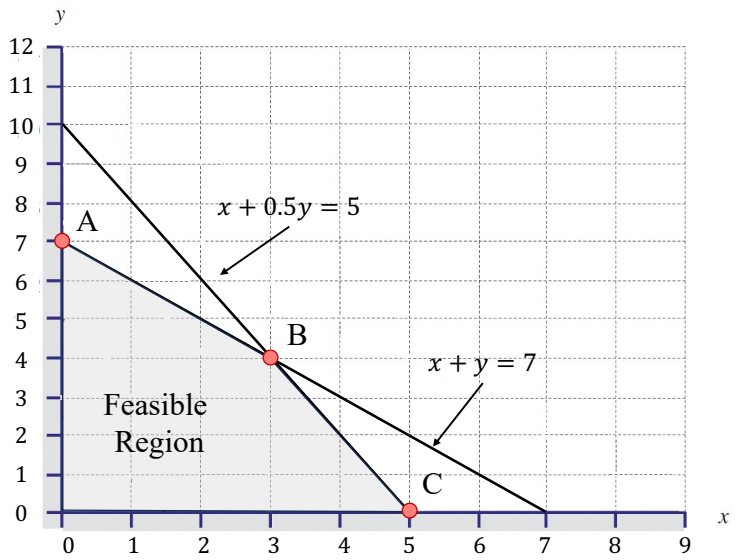
(a) (**3 pts**) Formulate the problem as a linear programming problem.

x = number of blouses to make

y = number of skirts to make

$$\begin{aligned} \text{maximize} \quad & P = 8x + 6y \\ \text{subject to} \quad & x + y \leq 7 \\ & x + 0.5y \leq 5 \\ & x \geq 0, y \geq 0 \text{ and } x, y \in \mathbb{Z} \end{aligned}$$

(b) (**3 pts**) Draw the constraints and identify the feasible region using the provided graph below.



(c) (**3 pts**) Solve the linear programming problem you defined in (a).
Maximum profit occurs at point B ($x = 3, y = 4$), and the profit is given by

$$P = 8 \times 3 + 6 \times 4 = 24 + 24 = 48$$