CSCE 580: Introduction to AI

Week 10 - Lectures 18 and 19: Adversarial Search; Constraints & Optimization

PROF. BIPLAV SRIVASTAVA, AI INSTITUTE

21ST AND 23RD OCT 2025

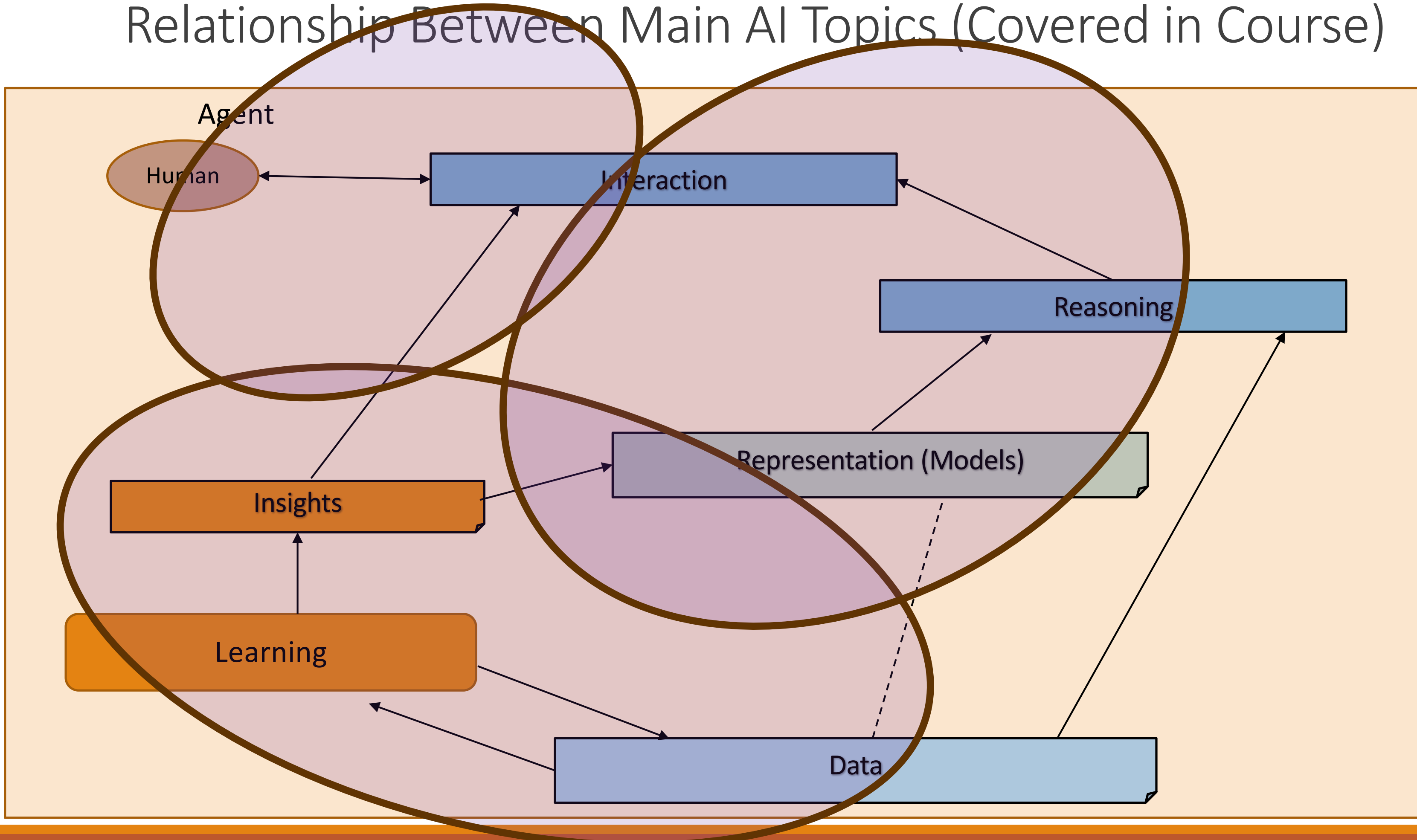
Carolinian Creed: "I will practice personal and academic integrity."

Credits: **Copyrights of all material reused acknowledged**

Organization of Week 10 - Lectures 18, 19

- Introduction Section
  - Recap from Week 9 (Lectures 16 and 17)
  - AI news
- Main Section
  - Lecture 18: Adversarial and Game Search
    - Games solving
    - Prisoner's Dilemma
  - Lecture 19: Optimization
    - Constraints
    - Optimization
- Concluding Section
  - About next week – W11: Lectures 20, 21
  - Ask me anything

Relationship Between Main AI Topics (Covered in Course)



Recap of Week 9

We discussed

- Informed search
- Local search

- Week 1: Introduction, Aim: Chatbot / Intelligence Agent
- Weeks 2-3: Data: Formats, Representation and the Trust Problem
- Week 3: Machine Learning – Supervised (Classification)
- Week 4: Machine Learning - Unsupervised (Clustering) –
- Topic 5: Learning neural network, deep learning, Adversarial attacks
- Week 6: Large Language Models – Representation and Usage issues
- Weeks 7-8: Search, Heuristics - Decision Making
- Week 9: Constraints, Optimization – Decision Making
- Topic 10: Markov Decision Processes, Hidden Markov models - Decision making
- Topic 11-12: Planning, Reinforcement Learning – Sequential decision making
- Week 13: Trustworthy Decision Making: Explanation, AI testing
- Week 14: AI for Real World: Tools, Emerging Standards and Laws; Safe AI/ Chatbots

Upcoming Evaluation Milestones

- Projects B: Sep 30 – Nov 20
- Quiz 2: Oct 7
- Quiz 3: Nov 11
- Paper presentation (grad students only) : Nov 18
- Finals: Dec 11

AI News

#1 NEWS – OpenAI's **Sora 2**

- Link: https://cdn.openai.com/pdf/50d5973c-c4ff-4c2d-986f-c72b5d0ff069/sora_2_system_card.pdf

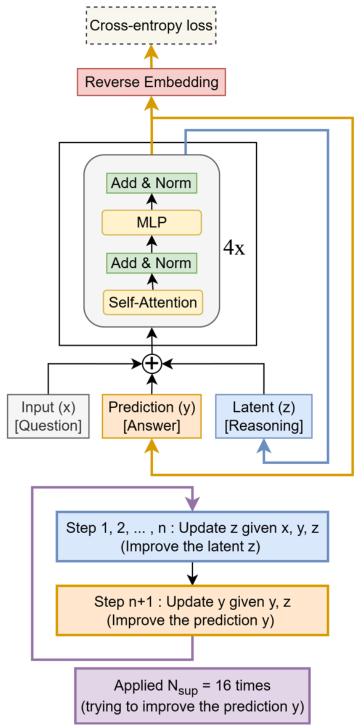
- Safety measures: Text and image moderation via multi-modal moderation classifiers: Input prompts, output video frames, audio transcripts, comments, and output scene description texts are run through various safety models
  - **Input (prompt) blocking**: This strategy involves blocking the tool from generating a video if text or image classifiers flag the prompt as violating our policies. By preemptively identifying and blocking inputs, this measure helps prevent the generation of disallowed content before it even occurs.
  - **Output blocking**: This approach, applied after the video has been generated, uses a combination of controls including Child Sexual Abuse Material (CSAM) classifiers and a safety-focused reasoning monitor to block the output of videos that violate our policies in the event that our input blocks have been circumvented
- Tightening usage policies
- Evaluation on safety: The production safety stack—scanning video frames, captions, and audio transcripts—was tested for two key metrics:
  - not_unsafe, measuring how effectively unsafe content is blocked (recall),
  - not_overrefuse, measuring how well benign content avoids false blocks.

Table 1: Safety Evaluations

Category	not_unsafe at output	not_overrefuse at output
Adult Nudity / Sexual Content Without Use of Likeness	96.04%	96.20%
Adult Nudity / Sexual Content With Use of Likeness	98.40%	97.60%
Self-Harm	99.70%	94.60%
Violence and Gore	95.10%	97.00%
Violative Political Persuasion	95.52%	98.67%
Extremism/Hate	96.82%	99.11%

#2 NEWS – Very Small Models for Game Solving

- TRM - Tiny Recursive Models: https://alexiajm.github.io/2025/09/29/tiny_recursive_models.html, https://arxiv.org/pdf/2510.04871
- HRM - Hierarchical Reasoning Model (HRM): https://github.com/sapientinc/HRM



Tiny Recursion Model (TRM) recursively improves its predicted answer y with a tiny network. It starts with the embedded input question x and initial embedded answer y and latent z. For up to K improvements steps, it tries to improve its answer y. It does so by

i) recursively updating n times its latent z given the question x, current answer y, and current latent z (recursive reasoning), and then

ii) updating its answer y given the current answer y and current latent z.

Table 3. % Test accuracy on Sudoku-Extreme dataset. HRM versus TRM matched at a similar effective depth per supervision step $(T(n+1)n_{layers})$

		HRM $n = k$, 4 layers		TRM $n = 2k$, 2 layers	
k	T	Depth	Acc (%)	Depth	Acc (%)
1	1	9	46.4	7	63.2
2	2	24	55.0	20	81.9
3	3	48	61.6	42	87.4
4	4	80	59.5	72	84.2
6	3	84	62.3	78	OOM
3	6	96	58.8	84	85.8
6	6	168	57.5	156	OOM

Introduction Section

Main Section

Lecture 18:
Adversarial and Game Search

Offline and Online Search

- All search studied until now were offline search
  - Solution found and then agent executed
- Online search
  - Interleave solution finding and execution
  - Cannot guarantee solution, unless actions have inverse-actions to undo state change

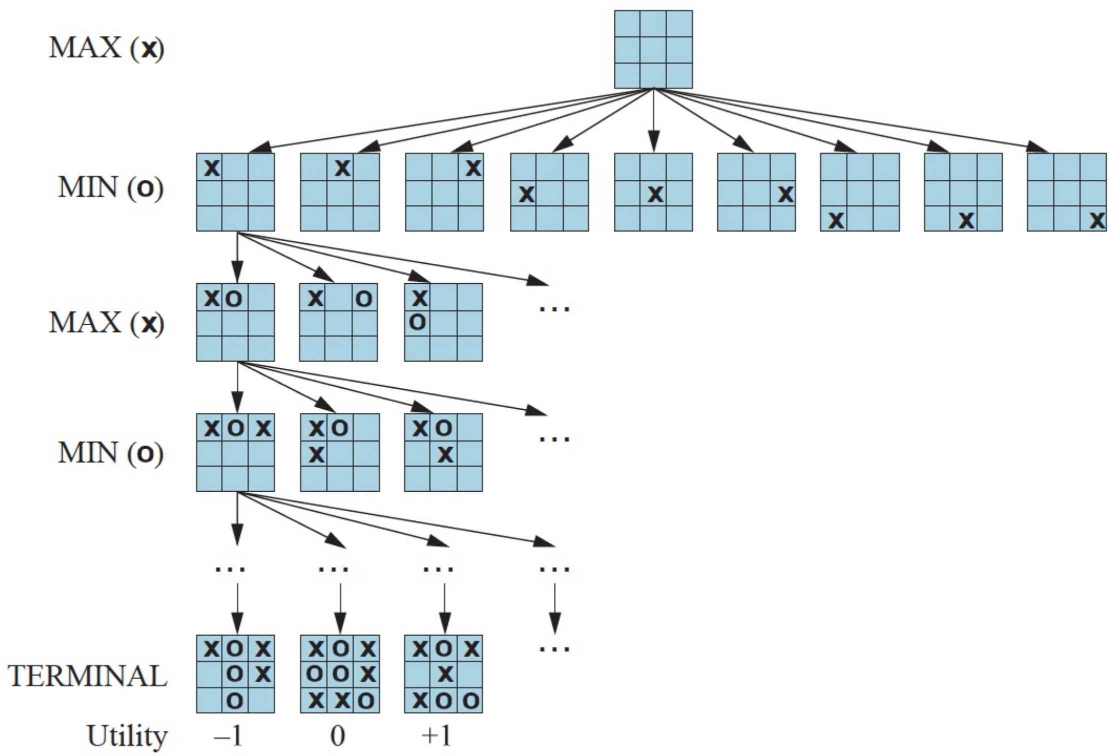
Games and Search

- Common setting
  - Two players
    - One human, one automated [**Common case**]
    - Both automated
  - Perfect information (fully observable)
  - Zero-sum – if one wins, another looses
- Captures many types of games
  - Tic-tac-toe, chess, Go, backgammon, …

Games and Search

- Setup
  - **So**: the initial state
  - **TO-MOVE(s)**: the player to play in state s
  - **ACTIONS(s)**: the set of legal actions at state s
  - **RESULT (s, a):** transition model
  - **IS-TERMINAL (s)**: terminal test to determine if game is over
  - **UTILITY (s, p):** utility or objective function. Numeric value capturing value of state s to player p

Tic-Tac-Toe – Game Tree

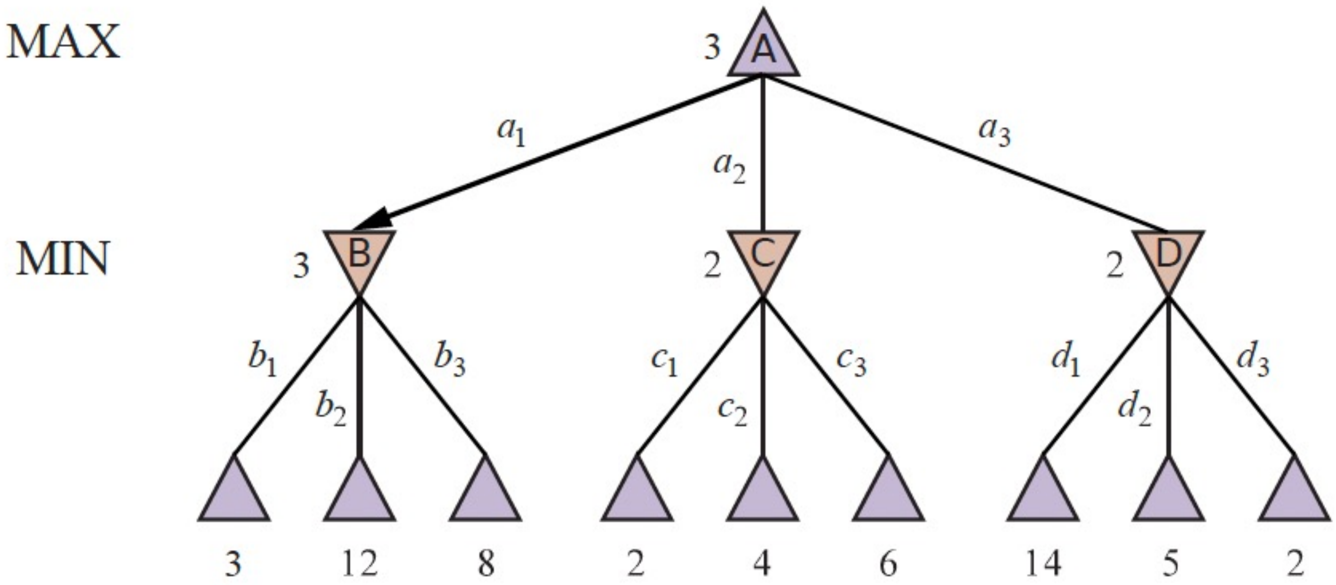


Alternate player turns

Adapted from:
Russell & Norvig, AI: A Modern Approach

MiniMax Utility

MINMAX(s) =
- UTILITY(s, MAX) if IS-TERMINAL(s)
- Max a in ACTIONS(S) MINIMAX(RESULT(s, a)) if TO-MOVE(s) = MAX
- Min a in ACTIONS(S) MINIMAX(RESULT(s, a)) if TO-MOVE(s) = MIN



Adapted from:
Russell & Norvig, AI: A Modern Approach

Minimax Search Algorithm

function MINIMAX-SEARCH(*game*, *state*) **returns** *an action*
  player ← *game*.TO-MOVE(*state*)
  value, *move* ← MAX-VALUE(*game*, *state*)
  return *move*

function MAX-VALUE(*game*, *state*) **returns** a (*utility*, *move*) pair
  if *game*.IS-TERMINAL(*state*) **then return** *game*.UTILITY(*state*, *player*), *null*
  $v \leftarrow -\infty$
  for each *a* **in** *game*.ACTIONS(*state*) **do**
    v2, *a2* ← MIN-VALUE(*game*, *game*.RESULT(*state*, *a*))
    if $v2 > v$ **then**
      v, *move* ← *v2*, *a*
  return *v*, *move*

function MIN-VALUE(*game*, *state*) **returns** a (*utility*, *move*) pair
  if *game*.IS-TERMINAL(*state*) **then return** *game*.UTILITY(*state*, *player*), *null*
  $v \leftarrow +\infty$
  for each *a* **in** *game*.ACTIONS(*state*) **do**
    v2, *a2* ← MAX-VALUE(*game*, *game*.RESULT(*state*, *a*))
    if $v2 < v$ **then**
      v, *move* ← *v2*, *a*
  return *v*, *move*

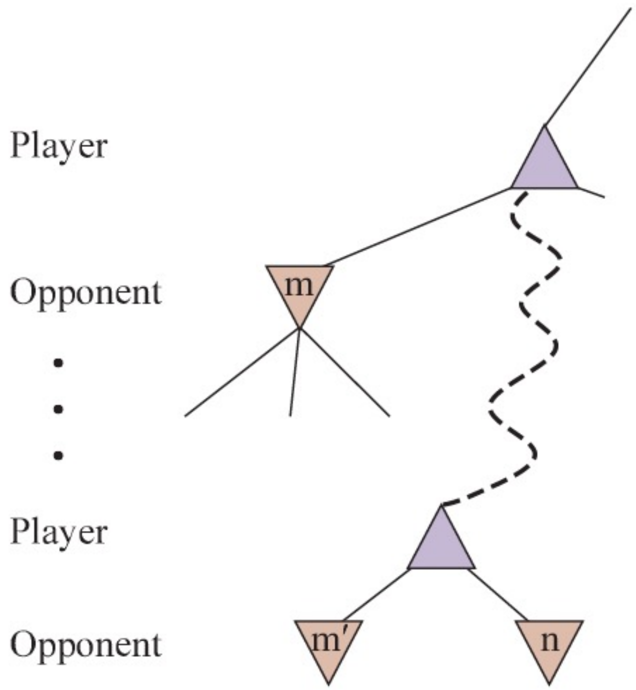
Starts from a Max node
Recursively does min on children,
who in-turn does max on children
to get value and corresponding
action

Adapted from:
Russell & Norvig, AI: A Modern Approach

Multi-player (>2) Games

- Possible to extend MINMAX, algorithm is more complex
  - Extend values with vectors, corresponding to per player
  - Levels increase (per turn)
- In real games, players can often form alliances
  - Hard to model

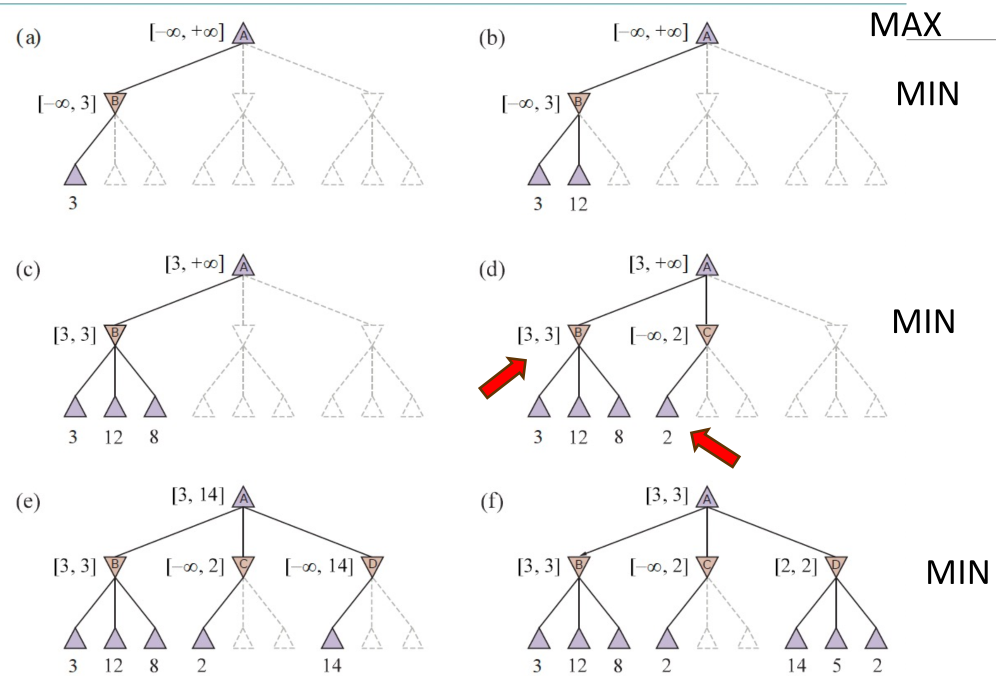
Alpha-Beta Pruning



If m or m′ is better than n for Player, search will never get to n

Adapted from:
Russell & Norvig, AI: A Modern Approach

Alpha-Beta Pruning



The first leaf below C has the value 2. Hence, C, which is a MIN node, has a value of at most 2. But we know that B is worth 3, so MAX would never choose C.

Adapted from:
Russell & Norvig, AI: A Modern Approach

Alpha-Beta Pruning

function ALPHA-BETA-SEARCH($game$, $state$) **returns** an action
  player ← $game$.TO-MOVE($state$)
  $value$, $move$ ← MAX-VALUE($game$, $state$, $-\infty$, $+\infty$)
  return $move$

function MAX-VALUE($game$, $state$, α, β) **returns** a ($utility$, $move$) pair
  if $game$.IS-TERMINAL($state$) **then return** $game$.UTILITY($state$, $player$), $null$
  $v \leftarrow -\infty$
  for each a **in** $game$.ACTIONS($state$) **do**
    $v2$, $a2$ ← MIN-VALUE($game$, $game$.RESULT($state$, a), α, β)
    if $v2 > v$ **then**
      v, $move$ ← $v2$, a
      $\alpha \leftarrow$ MAX(α, v)
    if $v \geq \beta$ **then return** v, $move$
  return v, $move$

function MIN-VALUE($game$, $state$, α, β) **returns** a ($utility$, $move$) pair
  if $game$.IS-TERMINAL($state$) **then return** $game$.UTILITY($state$, $player$), $null$
  $v \leftarrow +\infty$
  for each a **in** $game$.ACTIONS($state$) **do**
    $v2$, $a2$ ← MAX-VALUE($game$, $game$.RESULT($state$, a), α, β)
    if $v2 < v$ **then**
      v, $move$ ← $v2$, a
      $\beta \leftarrow$ MIN(β, v)
    if $v \leq \alpha$ **then return** v, $move$
  return v, $move$

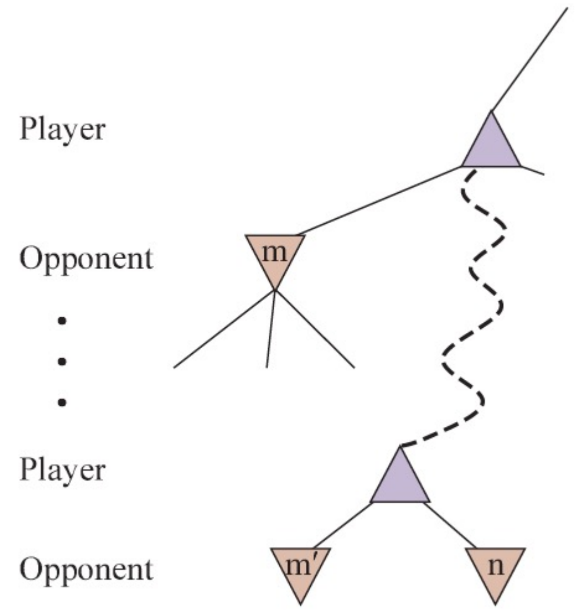
Alpha: the best (highest, at least) value
Beta: the best (lowest, at most) value

Adapted from:
Russell & Norvig, AI: A Modern Approach

Impact of Alpha Beta

- MINIMAX(O (b ^ m)

- Cuts down on size of game tree searched
  - Reduction depends on order of nodes traversed
  - Assuming order of nodes is random, expect to prune after exploring half the branches
  - Best case: O (b ^ (m/2)), where b is branching factor and m is depth



Game Setting

- Gaming strategies
  - Random
  - Minimax
  - alphabeta
- two automated players
  - First: random
  - Second: alphabeta
  - Code Example: https://github.com/aimacode/aima-python/blob/master/games4e.ipynb
- one human, one automated
  - First: human
  - Second: minimax

Heuristic Alpha Beta Tree Search

H-MINIMAX(s) =
- Eval(s, MAX) if IS-CUTOFF(s, d)
- Max a in ACTIONS(S) H-MINIMAX(RESULT(s, a), d+1) if TO-MOVE(s) = MAX
- Min a in ACTIONS(S) H-MINIMAX(RESULT(s, a), d+1) if TO-MOVE(s) = MIN

* Cut-off search at depth d
* Estimate utility of a state to player just like a heuristic function

UTILITY(loss, p) <= EVAL (s, p) <= UTILITY(win, p)

Useful when
- Depth is very high, example Chess
- Good (informative) eval functions exists

Adapted from:
Russell & Norvig, AI: A Modern Approach

Coding Example

- 2-party games code notebook
  - https://github.com/aimacode/aima-python/blob/master/games4e.ipynb
  - Has
    - Minimax
    - Alphabeta
    - Heuristic alpha beta

Searching in Larger Games

Game characteristics

- large branching factor – Go (361)
- difficult to get a meaningful evaluation function (heuristics)

Key Idea

- Value of a state is estimated as the average utility over a number of simulations of complete games from that state
- Get information of good plays by self-play and learning using neural networks

Monte Carlo Tree Search (MTCS)

```
function MONTE-CARLO-TREE-SEARCH(state) returns an action
  tree ← NODE(state)
  while IS-TIME-REMAINING() do
    leaf ← SELECT(tree)
    child ← EXPAND(leaf)
    result ← SIMULATE(child)
    BACK-PROPAGATE(result, child)
  return the move in ACTIONS(state) whose node has highest number of playouts
```

Adapted from:
Russell & Norvig, AI: A Modern Approach

MCTS Selection

- Selection: “upper confidence bounds applied to trees” (UCT)
  - U(n) is the total utility of all playouts that went through node n,
  - N(n) is the number of playouts through node n,
  - PARENT(n) is the parent node of n in the tree.

Credit: AIMA Book, 4th edition
AI – a Modern Approach

$$UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(\textsc{Parent}(n))}{N(n)}}$$

- Comments
  - N(n) is the exploitation term: the average utility of n.
  - The term with the square root is the exploration term: it has the count N(n) in the denominator, which means the term will be high for nodes that have only been explored a few times.

Monte Carlo Tree Search (*MCTS*)

```
function MONTE-CARLO-TREE-SEARCH(state) returns an action
  tree ← NODE(state)
  while IS-TIME-REMAINING() do
    leaf ← SELECT(tree)
    child ← EXPAND(leaf)
    result ← SIMULATE(child)
    BACK-PROPAGATE(result, child)
  return the move in ACTIONS(state) whose node has highest number of playouts
```

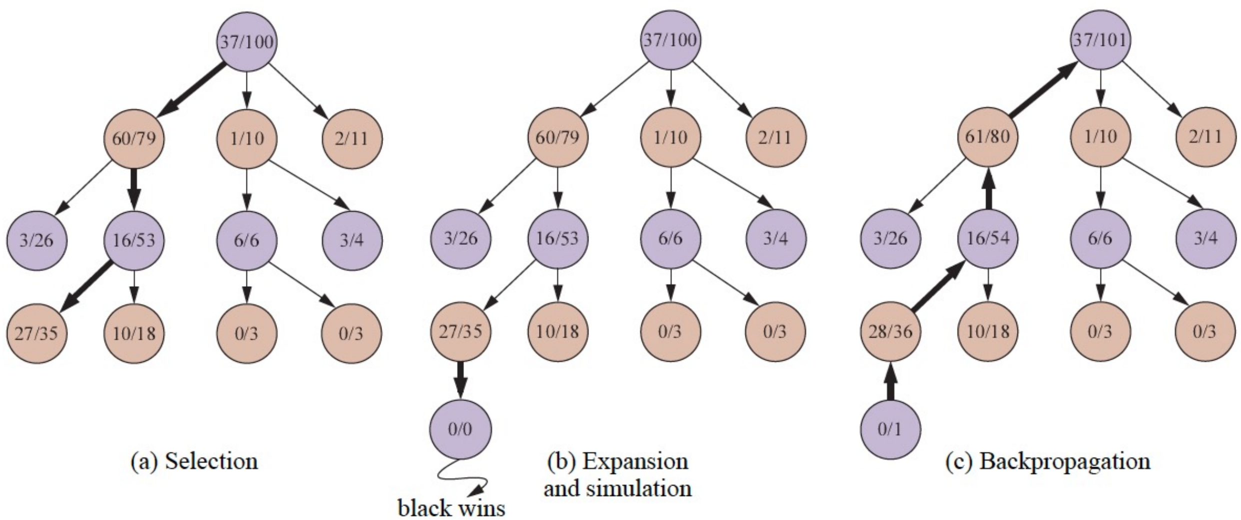


Figure 5.10 One iteration of the process of choosing a move with Monte Carlo tree search (MCTS) using the upper confidence bounds applied to trees (UCT) selection metric, shown after 100 iterations have already been done. In (a) we select moves, all the way down the tree, ending at the leaf node marked 27/35 (for 27 wins for black out of 35 playouts). In (b) we expand the selected node and do a simulation (playout), which ends in a win for black. In (c), the results of the simulation are back-propagated up the tree.

Adapted from:
Russell & Norvig, AI: A Modern Approach

Coding Example

- MCTS code notebook
  - https://pypi.org/project/mcts-simple/
  - https://colab.research.google.com/drive/1uYCDn_IymEhexepKfBXcMqiHquyhZpZ5?usp=sharing

References:
- https://deepmind.google/technologies/alphago/
- https://deepmind.google/technologies/alphazero-and-muzero/
- https://en.wikipedia.org/wiki/AlphaGo
- https://deepmind.google/discover/blog/alphastar-grandmaster-level-in-starcraft-ii-using-multi-agent-reinforcement-learning/

Alpha Solvers

- AlphaGo
  - Search with neural network based estimation
  - Two NNs: first - “policy network” — selects the next move to play, second — the “value network” — predicts the winner of the game.
  - Developed for Go; learned from human players and self-play
- AlphaZero
  - Given rules of the game (in mathematical form)
  - Uses reinforcement learning
  - Plays multiple games (Go, Chess, Shogi), learned from self-play
  - Developed newer methods for faster sorting, hashing, and matrix multiplication algorithms
- MuZero
  - MuZero is an advancement over AlphaZero, as it can learn the rules of a game or environment itself
  - Models three aspects of its environment - how good is the current position? Which action is the best to take? And how good was the last action?
  - Plays multiple games (Go, Chess, Shogi, Atari family), learned from self-play
- AlphaDev
  - Self-play via reinforcement learning, multi-agent learning, and imitation learning
  - Specialized MuZero system to discover more efficient computer science algorithms
  - Example: self-play with a group of players (league)

Real World Two+ Party Games – Wars, Tariffs ..

- Players
  - Countries, blocks (teams)
- Moves / actions
  - Economy: Buy, sell, hold, create and break treaties
  - Wars: attack, defend, stay-neutral, create and break treaties
- Rewards
  - Economy: market shares, profits, …
  - Wars: territories, battle losses, …
- Simulators
  - Victoria 3: https://www.thegamer.com/victoria-3-market-management-import-export-convoys-tariffs/

Two Party Decisions - Games

Prisoner's dilemma

- Two prisoners are caught for a robbery. They can testify against each other (-5 years to other; 0 themselves), stay silent (-10 year if other testifies, but -1 if they do not).
- For A: testifying (defecting) is a better choice (- 0 - 5 * ½) = -2.5 over remaining silent (cooperating) (-1 -10 * ½) = -6.5 // Assuming B will decide with probability 0.5
- For B: similarly, testifying is better
- For both, cooperating is better: -1 each, but the authorities would try to prevent it

Prisoner B / **Prisoner A**	**Prisoner B stays silent (*cooperates*)**	**Prisoner B testifies (*defects*)**
Prisoner A stays silent (*cooperates*)	Each serves 1 year	Prisoner A: 10 years Prisoner B: goes free
Prisoner A testifies (*defects*)	Prisoner A: goes free Prisoner B: 10 years	Each serves 5 years

Lecture 18: Summary

- We talked about
  - Games and Search
  - Minimax
  - Alphabeta
  - Monte Carlo Tree Search

Lecture 19: Optimization

Lecture 19: Outline

We will discuss

- Constraints Satisfaction Problems
- Optimization

Constraint Satisfaction Problems (CSPs)

- X - A set of variables $\{X_1, \ldots, X_n\}$
- D - A set of domains $\{D_1, \ldots, D_n\}$, for each variable
- C – set of constraints specifying allowed combinations of values for variables

Example

- $X_1 = \{1,2,3\}$, $X_2 = \{1,2,3\}$
  - Here, $D_1 = D_2 = \{1,2,3\}$
- C = < (X_1, X_2), $X_1 > X_2$ >

Solutions = Assignments to $(X_1, X_2) = \{(3,1), (3,2), (2,1)\}$

Example: Map-Coloring

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

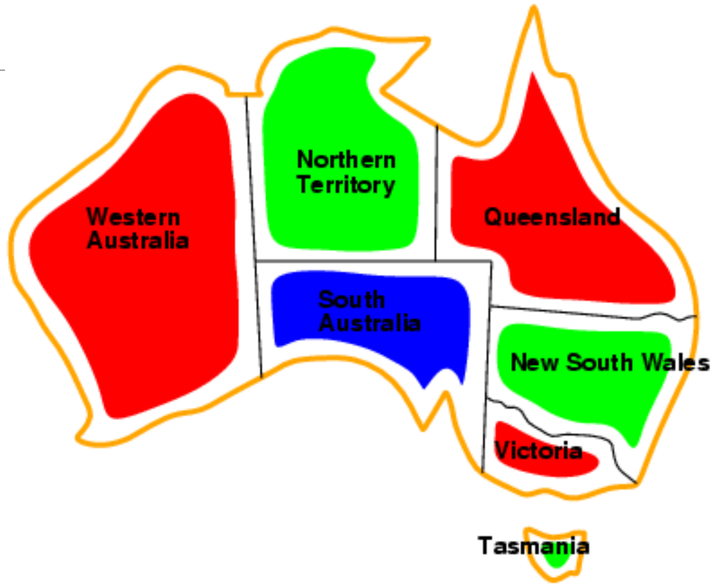
Domains D_i = {red, green, blue}

Constraints: adjacent regions must have different colors

e.g., WA ≠ NT, or (WA,NT) in {(red,green),(red,blue),(green,red),(green,blue),(blue,red),(blue,green)}

Adapted from:
Tuomas Sandholm's CSP Lecture
Russell & Norvig, AI: A Modern Approach

Example: Map-Coloring



Solutions are complete and consistent assignments

e.g., WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue,T = green

Adapted from:
Tuomas Sandholm's CSP Lecture
Russell & Norvig, AI: A Modern Approach

Varieties of CSPs

Discrete variables

- finite domains:
  - n variables, domain size $d \rightarrow O(d^n)$ complete assignments
  - e.g., Boolean CSPs, includes Boolean satisfiability (NP-complete)
- infinite domains:
  - integers, strings, etc.
  - e.g., job scheduling, variables are start/end days for each job
  - need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$

Continuous variables

- e.g., start/end times for Hubble Space Telescope observations
- linear constraints solvable in polynomial time by LP

Adapted from:
Tuomas Sandholm's CSP Lecture
Russell & Norvig, AI: A Modern Approach

Varieties of constraints

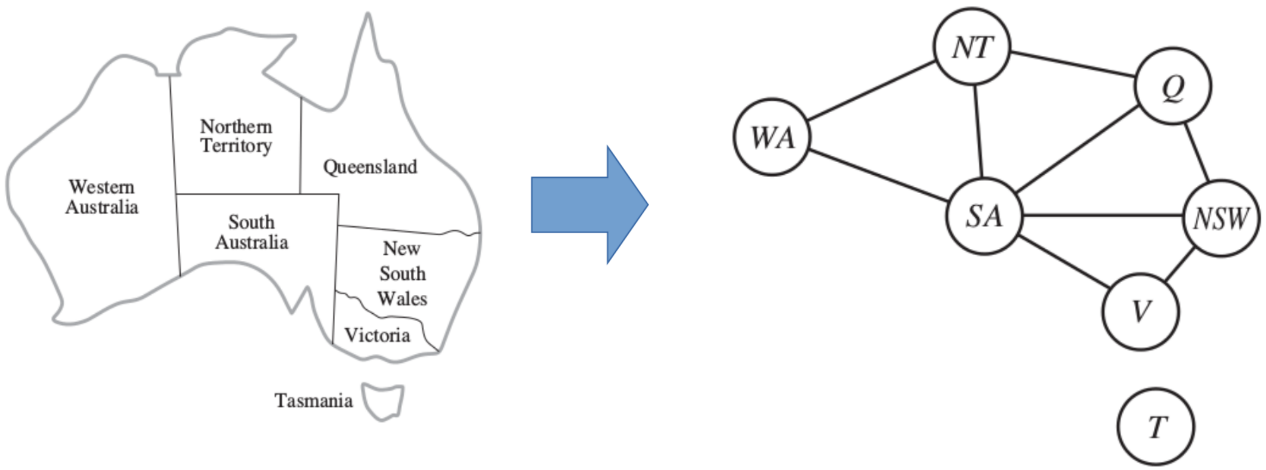
Unary constraints involve a single variable,
- e.g., SA ≠ green

Binary constraints involve pairs of variables,
- e.g., SA ≠ WA

Higher-order constraints involve 3 or more variables,
- e.g., cryptarithmetic column constraints

Adapted from:
Tuomas Sandholm's CSP Lecture
Russell & Norvig, AI: A Modern Approach

The constraint graph



Binary CSP: each constraint relates at most two variables

Constraint graph: nodes are variables, arcs show constraints

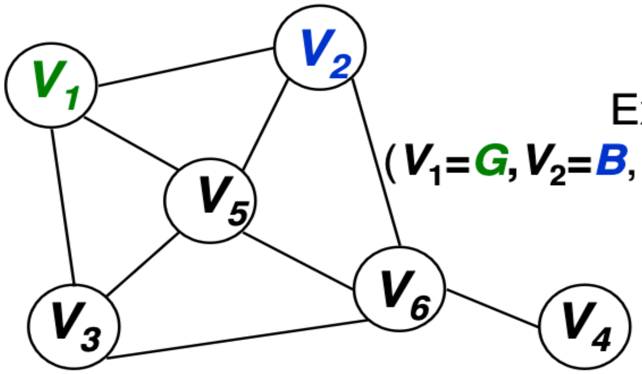
General-purpose CSP algorithms use the graph structure to speed up search

E.g., Tasmania is an independent subproblem!

Adapted from:

- https://www.khoury.northeastern.edu/home/camato/5100/csp.pdf
- https://www.cs.cmu.edu/afs/cs/academic/class/15381-s07/www/slides/020107CSP.pdf

Search Space



Example state:
$(V_1 = G, V_2 = B, V_3 = ?, V_4 = ?, V_5 = ?, V_6 = ?)$

- *State*: assignment to k variables with $k+1,..,N$ unassigned
- *Successor*: The successor of a state is obtained by assigning a value to variable $k+1$, keeping the others unchanged
- *Start state*: $(V_1=?, V_2=?, V_3=?, V_4=?, V_5=?, V_6=?)$
- *Goal state*: All variables assigned with constraints satisfied
- No concept of cost on transition → We just want to find a solution, we don't worry how we get there

Adapted from:
- Tuomas Sandholm's CSP Lecture
- https://www.khoury.northeastern.edu/home/camato/5100/csp.pdf
- https://www.cs.cmu.edu/afs/cs/academic/class/15381-s07/www/slides/020107CSP.pdf

Example: Cryptarithmetic

- Variables

  D, E, M, N, O, R, S, Y
- Domains

  {0, 1, 2, 3 ,4, 5, 6, 7, 8, 9 }
- Constraints

  $M \neq 0$, $S \neq 0$ (unary constraints)

  $Y = D + E$ OR $Y = D + E - 10$.

  $D \neq E$, $D \neq M$, $D \neq N$, etc.

$$\begin{array}{r} \text{S E N D} \\ \underline{+\text{M O R E}} \\ \text{M O N E Y} \end{array}$$

Adapted from:

- Tuomas Sandholm's CSP Lecture
- https://www.khoury.northeastern.edu/home/camato/5100/csp.pdf
- https://www.cs.cmu.edu/afs/cs/academic/class/15381-s07/www/slides/020107CSP.pdf

A harder CSP to represent: Cryptarithmetic

- **Variables:**

$$F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$$

- **Domains:**

$$\{0,1,2,3,4,5,6,7,8,9\}$$

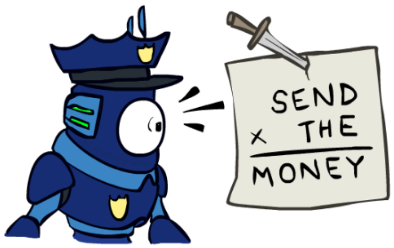
- **Constraints:**

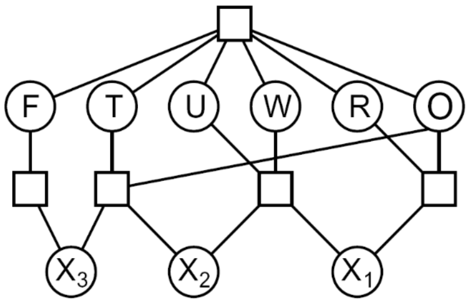
$$\text{alldiff}(F,T,U,W,R,O)$$

$$O+O=R+10\cdot X_1$$

$$\ldots$$

```
    T W O
+   T W O
---------
  F O U R
```

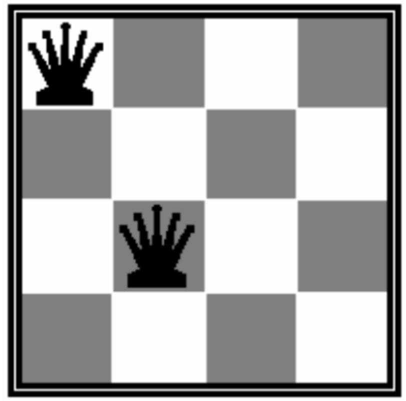




Adapted from:
- https://www.khoury.northeastern.edu/home/camato/5100/csp.pdf
- https://www.cs.cmu.edu/afs/cs/academic/class/15381-s07/www/slides/020107CSP.pdf

Example: N-Queens

- Variables: Q_i
- Domains: $D_i = \{1, 2, 3, 4\}$
- Constraints
  - $Q_i \neq Q_j$ (cannot be in same row)
  - $|Q_i - Q_j| \neq |i - j|$ (or same diagonal)



$Q_1 = 1 \quad Q_2 = 3$

- Valid values for (Q_1, Q_2) are (1,3) (1,4) (2,4) (3,1) (4,1) (4,2)

Adapted from:
- Tuomas Sandholm’s CSP Lecture
- https://www.khoury.northeastern.edu/home/camato/5100/csp.pdf
- https://www.cs.cmu.edu/afs/cs/academic/class/15381-s07/www/slides/020107CSP.pdf

Constraint Propagation

- **Node Consistency**: a variable (node in CSP graph) is node—consistent of all the values in the variable's domain satisfy variable's unary constraints
- **Arc Consistency**: every variable in its domain satisfies binary constraints

function AC-3(csp) **returns** false if an inconsistency is found and true otherwise
  $queue \leftarrow$ a queue of arcs, initially all the arcs in csp

  while $queue$ is not empty **do**
    $(X_i,\ X_j) \leftarrow$ POP($queue$)
    if REVISE(csp, X_i, X_j) **then**
      if size of $D_i = 0$ **then return** $false$
      for each X_k **in** X_i.NEIGHBORS - $\{X_j\}$ **do**
        add $(X_k,\ X_i)$ to $queue$
  return $true$

function REVISE(csp, X_i, X_j) **returns** true iff we revise the domain of X_i
  $revised \leftarrow false$
  for each x **in** D_i **do**
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j **then**
      delete x from D_i
      $revised \leftarrow true$
  return $revised$

Adapted from:
Russell & Norvig, AI: A Modern Approach

Constraint Propagation

- **Path Consistency**: A two variables set $\{X_i , X_j\}$ is path-consistent with respect to a third variable X_m if, for every assignment $\{X_i = a_i, X_j = a_j\}$ consistent with constraints on $\{X_i , X_j\}$, there is an assignment to X_m which is consistent with $\{X_i , X_m\}$ and $\{X_m , X_j\}$

- **k-consistency**: A CSP is k-consistent if for any set of (k-1) variables and their consistent assignments, a consistent value can always be assigned for the k^{th} variable.

CSP: Coding Example

- CSP code notebook
  - https://github.com/aimacode/aima-python/blob/master/csp.ipynb/

Making Optimal Decisions

Optimal Decision

- What is it? There is no absolute answer. In AI, there is the concept of a **rational** agent.
- Acting rationally: acting such that one ca achieve one's goals given one's beliefs (and information)
  - But what are one's goals
  - Are the always of achievement?
- Some options
  - Perfect rationality: maximize expected utility at every time instant
    - Given the available information; can be computationally expensive
    - "Doing the right thing"
  - Bounded optimality: do as well as possible given computational resources
    - Expected utility as high as any other agent with similar resources
  - Calculative rationality: *eventually* returns what would have been the rational choice

What Is It?

- As a working principle
  - Bounded or Calculative Rationality
- In observable and deterministic scenarios
  - Maximize utility: (benefit – cost)
- In scenarios with uncertainty and/ or unobservable
  - Maximize ***expected*** utility: (benefit – cost)

Example Situation – Course Selection

- A person wants to pass an academic program in two majors: A and B
- There are three subjects: A, B and C, each with three levels (*1, *2, *3). There are thus 9 courses: A1, A2, A3, B1, B2, B3, C1, C2, C3
- To graduate, at least one course at beginner (*1) level is needed in major(s) of choice(s), and two courses at intermediate levels (*2) are needed
- **Optimality considerations** in the problem
  - Least courses, fastest time to graduate, class size, friends attending together, …
- Answer questions
  - Q1: How many minimum courses does the person have to take ?
  - Q2: Can a person graduate in 2 majors studying 3 courses only?
  - …

Algorithms for Optimality

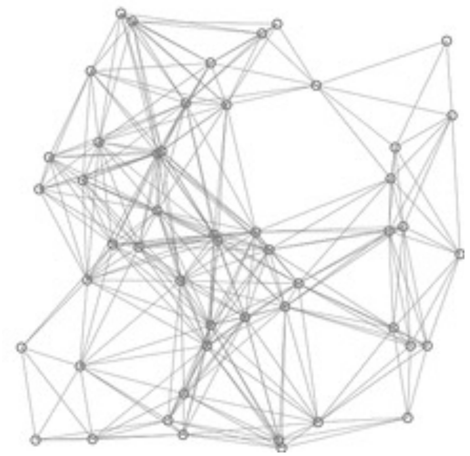
- Problem specific methods
  - Path finding
  - Linear programming
  - Constraint satisfaction and optimization

- General-purpose methods for optimality using search

Optimality: Example - Path Finding

Main steps

1. Mark all nodes unvisited. Create a set of all the unvisited nodes called the *unvisited set*.
2. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes. Set the initial node as current.
3. For the current node, consider all of its unvisited neighbors and calculate their *tentative* distances through the current node. Compare the newly calculated *tentative* distance to the current assigned value and assign the smaller one.
4. When we are done considering all of the unvisited neighbors of the current node, mark the current node as visited and remove it from the *unvisited set*. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the *unvisited set* is infinity, then stop. The algorithm has finished.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.



A demo of Dijkstra's algorithm based on Euclidean distance

Djikstra's Algorithm with positive numbers or labels that are [monotonically](https://en.wikipedia.org/wiki/Monotonic_function) non-decreasing.

Source: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Illustrating Optimality – Informed Search v/s Pathfinding

Online example site (seen in W9):

https://www.movingai.com/SAS/ASM/

- Djikstra's
- Weighted A*

Exercise and Code

- Linear Programming Methods – Google's OR-Tool
  - Link - https://github.com/biplav-s/course-d2d-ai/blob/main/sample-code/l16-optimal/Optimization.ipynb

OR and N- Queens

- Constraint programming-based optimization - CP tool
- Code sample: https://developers.google.com/optimization/cp/queens

Lecture 19: Summary

- We talked about
  - Constraint Satisfaction Problem
  - Optimization Problems
    - Different type of solvers

Week 10: Concluding Comments

We talked about

- Lecture 18: Adversarial and Game Search
- Lecture 19: Optimization

- Week 1: Introduction, Aim: Chatbot / Intelligence Agent
- Weeks 2-3: Data: Formats, Representation and the Trust Problem
- Week 3: Machine Learning – Supervised (Classification)
- Week 4: Machine Learning - Unsupervised (Clustering) –
- Topic 5: Learning neural network, deep learning, Adversarial attacks
- Week 6: Large Language Models – Representation and Usage issues
- Weeks 7-8: Search, Heuristics - Decision Making
- Week 9: Constraints, Optimization – Decision Making
- Topic 10: Markov Decision Processes, Hidden Markov models - Decision making
- Topic 11-12: Planning, Reinforcement Learning – Sequential decision making
- Week 13: Trustworthy Decision Making: Explanation, AI testing
- Week 14: AI for Real World: Tools, Emerging Standards and Laws; Safe AI/ Chatbots

Projects B: Sep 30 – Nov 20 (7 weeks; 400 points)

- End date: **Thursday, Nov 20**
  - Remember to update spreadsheet on data/ time when finished **(Column I)**

- Choices
  - Given by instructor
  - Defined by student using project-b teamplate; reviewed and approved by instructor

Upcoming Evaluation Milestones

- Projects B: Sep 30 – Nov 20
- Quiz 2: Oct 7
- Quiz 3: Nov 11
- Paper presentation (grad students only) : Nov 18
- Finals: Dec 11

About Week 11 – Lectures 20, 21

Week 11 – Lectures 20, 21

- Lecture 20: Making Decisions - Simple
- Lecture 21: Making Decisions - Complex

- Week 1: Introduction, Aim: Chatbot / Intelligence Agent
- Weeks 2: Data: Formats, Representation, ML Basics
- Week 3: Machine Learning – Supervised (Classification)
- Week 4: Machine Learning - Unsupervised (Clustering) –
- Topic 5: Learning neural network, deep learning, Adversarial attacks
- Week 6: Large Language Models – Representation and Usage issues
- Weeks 7-8: Search, Heuristics - Decision Making
- Week 9: Constraints, Optimization – Decision Making
- Topic 10: Markov Decision Processes, Hidden Markov models - Decision making
- Topic 11-12: Planning, Reinforcement Learning – Sequential decision making
- Week 13: Trustworthy Decision Making: Explanation, AI testing
- Week 14: AI for Real World: Tools, Emerging Standards and Laws; Safe AI/ Chatbots

Note: exact schedule changes slightly to accommodate for exams and holidays.