1 Agents and Abstraction

Definition 1.1 (Planning Horizon):

- **Static:** The world does not change over time.
- **Finite Horizon:** The agent reasons about a fixed finite number of time steps. (Agent know when it will end)
- **Indefinite Horizon:** The agent reasons about a finite but not predetermined number of time steps, such as until goal completion. (Agent know it will end, but not sure when)
- **Infinite Horizon:** The agent plans as if it will continue operating forever. (agent know it will never end)

Definition 1.2 (Representation):

- **Explicit States:** A state represents one possible configuration of the world.
- **Features:** Natural descriptors of states. (binary features can represent exponentially many states)
- **Individuals and Relations:** Use feature for reasoning about individuals and their relationships without necessarily knowing all individuals or when there are infinitely many individuals.

Definition 1.3 (Computational Limits):

- **Perfect Rationality:** The agent always selects the optimal action, which is often not possible in practice.
- **Bounded Rationality:** The agent selects a possibly sub-optimal action given its limited computational resources.

Definition 1.4 (Uncertainty):

- **fully observable:** agent knows the state of the world from the observation
- **partially observable:** there can be many state that are possible given an observation.

Definition 1.5 (Uncertain Dynamics):

- **Deterministic:** The outcome of an action is always the same.
- **Stochastic:** There is uncertainty over the states resulting from executing a given action.

Definition 1.6 (Goals or Complex Preferences):

- **Achievement Goals:** Goals that an agent aims to achieve, which can be represented as complex logical formulas.
- **Maintenance Goals:** Goals that an agent seeks to maintain over time.
- **Complex Preferences:** Involves trade-offs between various desiderata, potentially at different times, and may be either ordinal or cardinal (e.g: medical).

Definition 1.7 (Reasoning by Number of Agents):

- **Single Agent Reasoning:** The agent assumes any other agents are part of the environment, focusing on individual goal achievement.
- **Adversarial Reasoning:** The agent considers another agent acting in opposition to its goals, common in competitive settings.
- **Multi-agent Reasoning:** The agent strategically reasons about the actions and goals of other agents, which may be cooperative, competitive, or independent.

Algorithm	**Frontier**	**Runtime**	**Space**	**halts?**
uninformed (no heuristic)				
Depth-First Search	LIFO	Exp / $O(b^m)$	Linear / $O(bm)$	No
Breadth-First Search	FIFO	Exp / $O(b^d)$	Exp / $O(b^d)$	Yes
Lowest-Cost-First Search	Lowest cost	Exp	Exp	Yes
Dijkstra's Algorithm*	Lowest cost	$O((V+E)\log V)$	$O(V^2)$	
Iterative-Deepening Search*	LIFO in FIFO [1]	Exp / $O(b^d)$	Linear / $O(bd)$	Yes [2]
informed (has heuristic)				
(Greedy) Best-First Search	Global min heuristic	Exp	Exp	No
Heuristic Depth-First Search	Local min heuristic (LIFO)	Exp / $O(b^m)$	Linear / $O(bm)$	No
A* Search	Lowest (cost + heuristic)	Exp	Exp	Yes

b is the branching factor
m is the maximum depth of the search tree
d is the depth of the shallowest goal node.

[1] a BFS but for every depth limit do a DFS
[2] Guaranteed to terminate at depth d

Algorithm	**Completeness**	**Optimality**
uninformed (no heuristic)		
Depth-First Search	No (fails for infinite cycle)	No (not considering all possibilities)
Breadth-First Search	Yes	Yes (if cost is uniform)
Lowest-Cost-First Search	Yes	Yes
Dijkstra's Algorithm*	Yes	Yes
Iterative-Deepening Search*	Yes (Same as BFS)	No (but guaranteed shallowest goal)
informed (has heuristic)		
(Greedy) Best-First Search	No (fails for infinite cycle)	No (from not considering cost of arc)
Heuristic Depth-First Search	No (fails for infinite cycle)	No (not considering all possibilities)
A* Search	Yes[1]	Yes[1]

[1] Assuming heuristic is **admissible**, branch factor is **finite**, and arc cost are bounded **above zero**
If h satisfies the **monotone restriction**, A^* with **multiple path pruning always finds the shortest path** to a goal.

2 Graph Search Algorithm

Definition 2.1 (Admissible) An **Admissible** heuristic **never overestimate** the cost from **any node to the goal**. An **Admissible** search algorithm returns an **optimal solution** if it exists.

Definition 2.2 (Monotone / Consistent) A heuristic function h satisfies the **monotone restriction** if $h(m) - h(n) \leq \textbf{cost}(m, n)$ **for every arc** $< m, n >$. (The heuristic of a path is always less than or equal to the true cost)

Monotonicity is like **admissibility** but between any two nodes.
So, a **consistent** heuristic is **admissible**, but a **admissible** heuristic is not necessarily **consistent**.

3 Adversarial Search(Minimax)

- ```
  function minimax (node, depth, maximizingPlayer) is
      if depth = 0 or node is a terminal node then
          return the heuristic value of node
      if maximizingPlayer then
          value := -inf
          for each child of node do
              value := max(value, minimax(child, depth - 1, FALSE))
          return value
      else (* minimizing player *)
          value := inf
          for each child of node do
              value := min(value, minimax(child, depth - 1, TRUE))
          return value
  ```

- **Suitable Type of Problem:**
  - Competitive two-person, zero-sum games.
  - Two players take turns to move and the one winner one loser.

- **Idea:**
  - Find **best option** for you on nodes you control (MAX)
  - Assumes opponent will take **worst option** for you on their node (MIN)
  - Recursively search leaf nodes and percolate optimal value upward

- **Pruning Methods:**
  - **Alpha-beta Pruning:**
    * Ignore portions of the search tree without losing optimality
    * doesn not change worst-case performance (Exp)
  - **Heuristic Pruning (Early Stopping):**
    * Heuristics are used to evaluate the potential of non-terminal states.
    * This method saves computational resources but may not always yield the optimal solution.

4 Higher level strategies

Search	**Search Complexity**	**Difficulty**	**Reason to Win**
Symmetric	b^n	Not able to construct backward on dynamically constructed graph	Choose between forward / backward search based on branching factor
Bidirectional	$2b^{\frac{k}{2}} << b^k$	Make sure frontiers meet	Searches forward and backward simultaneously, leading to exponential savings in time and space.
Island-Driven	$mbk^{\frac{k}{m}} << b^k$	identify islands hard to guarantee optimality	Decomposes the problem into m smaller subproblems, each of which is easier to solve.
  ```

5 Constraint Satisfaction Problems (CSPs)

- **Definition:**
  - Set of **variables**, **domain for each variable**, set of **constraints** or **evaluation function**.
  - **Solution** is an assignment to the **variables** that satisfies **all constraints**.
  - **Solution** is a **model of the constraints**.
- **Problem Types:**
  - **Satisfiability Problems:** Find assignment satisfies the given hard constraints.
  - **Optimization Problems:** Find assignment optimizes the evaluation function(soft constraints).
- **Search Representation:**

Assignment Type	**Description**
Complete Assignment	Node is assignment of value to all variables. Neighbours are created by changing one variable value.
Partial Assignment	Nodes is assignment to the first $k-1$ variables. Neighbours are formed by assigning a value to the k^{th} variable.

- **Dual Representations of Crossword Puzzle:**

Type	**Nodes**	**Domains**	**Constraints**
Primal	word positions	letters	intersecting letters are same
Dual	squares	letters	words must fit

- **Example of CSP Setup:**

Problem	**Variables**	**Domains**	**Constraints**
Crosswords	letters	a-z	words in dictionary
Crosswords	words	dictionary	letters match
Scheduling	times events resources	times,dates types values	before, after same resource
Party Planning	guests	values	cliques
Ride Sharing	people/trips	locations	cars

- **Constraints & Solution**
  - **Constraints**: Can be **N-ary** or **Binary**.
  - **Solutions**:

    Generate and Test

    Exhaustively check all combinations against constraints.

    Backtracking

    Prune large portions of the state space by ordering variables and evaluating constraints.

    Efficiency depends on **order** of variables.

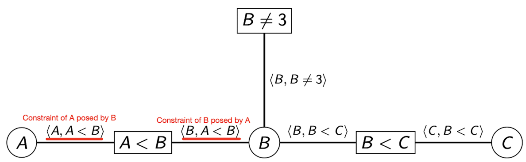
    Find optimal ordering is **as hard** as solving the problem.

    Cut off large branches as soon as possible, push failures as high as possible.

    Consistency Techniques

    Look for inconsistencies to simplify the problem / Graphical representation

- **Constraint Network (CN)**
  - **Domain constraint**:
    unary constraint of values x on values in a domain, $< X, c(X) >$
  - **Domain consistent**:
    A **node** is **Domain consistent** if no domain value violates any domain constraints.
    A **CN** is **Domain consistent** if all nodes are **Domain consistent**.
  - **Arc** $< X, c(X,Y) >$ is:
    A **constraint** on X posed by Y.
    Arc consistent if for all $X \in D_X$, there exist some $Y \in D_Y$ such that $c(X,Y)$ is satisfied.
  - **CN** is **Arc consistent** if all arcs are **arc consistent**.
  - set of variables $\{X_1, X_2, \ldots, X_N\}$ is **path consistent** if all **arcs** and **domains** are **consistent**

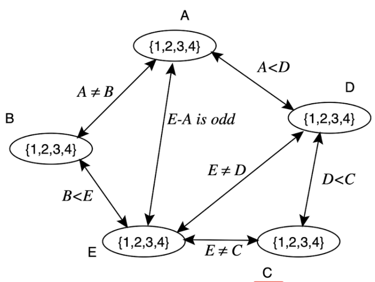


- **AC-3 (CN) algorithm** (Alan Mackworth, 1977)
  - **Purpose:** Makes a Constraint Network (CN) arc consistent and domain consistent.
  - **Procedure:**
    * Initialize the To-Do Arcs Queue (TDA) with all inconsistent arcs.
    * Make all domains domain consistent.
    * Put all arcs in TDA.
    * Repeat until TDA is empty:
      · Select and remove an arc $< X, c(X,Y) >$ from TDA.
      · Remove all values from the domain of X that:
        do not have a corresponding value in the domain of Y
        satisfying the constraint $c(X,Y)$.
      · If any values were removed, for all $Z \neq Y$, add back arcs $< Z, c'(Z,X) >$ into TDA . (Add back all constraints posed to other variable by X. As the X value enforced by the constraints / arc we removed is used by some other constraints posted by X to other variables)
  - **Termination:**
    * AC-3 always terminates under one of three conditions:
      · **Every domain is empty**: no solution.
      · **Every domain has a single value**: a solution.
      · **Some domains have 1+ value**: not sure if a solution exists.
        (further splitting and recursive call needed)
  - **Properties:**
    * Termination is guaranteed.
    * Time complexity is $O(cd^3)$.[1]
    * Consistency of each arc can be checked in $O(d^2)$ time.
  - **Different elimination ordering** can result in **different size** of intermediate constraints.

[1] where n is the number of variables, c is the number of binary constraints, and d is the maximum size of any domain

- **Variable Elimination**
  - **Concept:**
    - Variables are eliminated one by one, transferring their constraints to neighbours.
    - A single remaining variable with no values indicates an **inconsistent** network.
  - **Algorithm:**
    - If only one variable remains, return the intersection of the unary constraints involving it.
    - Select a variable X.

      Join the constraints where X appears to form a new constraint R.
      Project R onto other variables to form R_2.
      Place new constraint R_2 between all variables previously connected to X.
      Remove X from the problem.
      Recursively solve the simplified problem.
      Return R joined with the solution from the recursive call.
  - Finding the optimal elimination ordering is as complex as the CSP itself.



$r_1 : C \neq E$	C	E
	3	2
	3	4
	4	2
	4	3

$r_2 : C > D$	C	D
	3	2
	4	2
	4	3

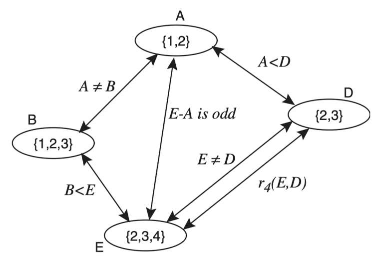
pi is the projection operator that projects R3 to D and E

$r_3 : r_1 \bowtie r2$	C	D	E
	3	2	2
All the value of C,D,E they can have together	3	2	4
	4	2	2
	4	2	3
	4	3	2
We are not consider the constraint between E and D i.e: E != D	4	3	3

$r_4 : \pi_{\{D,E\}} r_3$	D	E
	2	2
Represents what C does to D and E.	2	3
	2	4
	3	2
	3	3

$\hookrightarrow$ new constraint

We can create a R_5 that enforces E!=D.



- **Local Search:** (Back to CSP as Search)
  - Maintain a variable assignment, select neighbors to improve heuristic value, and stop when a satisfying assignment is found, or return the best assignment found.
  - Aim is to find an assignment with zero **unsatisfied constraints (Conflict)**
  - Goal is an assignment with **zero conflicts** (heuristic: # of conflicts)
- **Greedy Descent Variants:**
  - At every step:
    (Select the variable-value pair that **minimize** # of conflicts)
    (Select a variable involved in the **most** # of conflicts, then a value **minimize** # of conflicts)
    (Select a variable involved in **any** conflicts, then a value **minimize** # of conflicts)
    (Select a variable **at random**, then a value **minimize** # of conflicts)
    (Select a variable and value **at random**, accept if **doesn't increase**[2] # of conflicts)

[2] Sometime accept even increase # of conflicts to escape local minimum

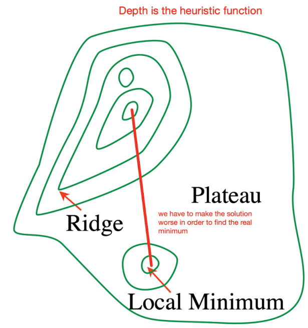
- **GSAT (Greedy SATisfiability):**
  - Start with a random assignment of values to all variables n
    heuristic $h(n) = \#$ of unsatisfied constraints
  - repeat until heuristic becomes 0 (Solved):

    Evaluate neighbours of n (not n, cannot change same variable twice in a row);

    Let n be the neighbour n' that minimizes the heuristic, even if $h(n') > h(n)$.
  - **Problems:**

    stuck at local minimum, cannot pass a **plateau** where $h(n)$ are uninformative.

    Ridge is a local minimum where **n-step look-ahead** might help.



  - **Randomized GSAT**: allow move to a random neighbour, reassign all variable randomly.
  - **Tabu lists**:

    Maintain a tabu list of k last assignments to prevent cycling.
    Reject assignments exist on tabu lists.[3]
    More efficient than a list of complete assignments, but expensive if k is large.

- **Stochastic local search** is a mix of:
  - **Greedy descent**: Move to a lowest neighbour (GSAT)
  - **Random walk**: taking some random steps
  - **Random restart**: reassigning all values randomly

- **Simulated Annealing** (Variant of Stochastic local research)
  - Pick a random value for a random variable (neighbour):
    Adopt if its an improvement.
    If it's not an improvement, adopt it probabilistically based on temperature T. (High $\rightarrow$ Low)
    (move from current assignment n to new assignment n' with probability $e^{-\frac{h(n')-h(n)}{T}}$)[4]
    Terminate when criteria is met.

- **Parallel Search:**
  - Maintains a population of k individuals (total assignments).
  - Updates each individual in the population at every stage.
  - Reports when any individual is a solution.
  - Operates like k restarts but with k times the minimum steps.

- **Beam Search:**
  - Similar to parallel search with k individuals, but **choosing the k best from all neighbours**.
    (All if there are less than k)
  - Reduces to greedy descent when $k = 1$.
  - The value of k limits space and facilitates parallelism.

[3] e.g: $k = 1$ means reject assignment of the same value to the variable chosen.
[4] difference in heuristic value divided by temperature

- **Stochastic Beam Search:**
  - A probabilistic variant of beam search, choosing k individuals for the next generation.
    (probability of a neighbour n is chosen is proportional to $e^{-\frac{h(n)}{T}}$)
  - Maintains diversity among individuals and reflects their fitness (heuristic).
  - Operates like asexual reproduction, with mutation allowing fittest individuals to prevail.
- **Genetic Algorithms:**
  - Like stochastic beam search but **combines pairs of individuals to create offspring**.
  - Fittest individuals are more likely to be chosen for reproduction.
  - Crossover and mutation (change some value) to form new solutions.
  - Continues until a solution is found.
- **Crossover:**
  - Given two individuals, each offspring's attributes are randomly chosen from one of the parents.
  - The effectiveness depends on the ordering of variables, many variations are possible.
- **Comparing Stochastic Algorithms:**
  - Considers how to compare algorithms with different success rates and runtimes.
  - Acknowledges that summary statistics like mean, median, or mode runtimes may not be informative.
- **Runtime Distribution:**
  - Plots runtime or steps against the proportion of runs solved within that time.
  - Helps in understanding the performance distribution of stochastic algorithms.

6 Inference and Planning

-