Uninformed Search

1. Which attributes are typically stored on the nodes of a search tree for uninformed search? What additional information would you need to store for informed search?

- Attributes typically stored on the nodes of a search tree:
  - A state
  - A link to the parent node which stores the last state before the current one
  - The last action taken before moving to this state
  - The depth of the node
  - The path cost from the root to the current node
- For informed search, we need all of the above plus:
  - Heuristic value, i.e the distance from the current node to the goal

2. What is the difference between TREE-SEARCH and GRAPH-SEARCH?

- tree search can visit a state multiple times

Related questions may include:

• Which type of search do we use when we are searching for a path on a grid? Why?

- :BFS. It's guarantees to gives the shortest path from the start to the goal, IF all edges have equal weight.

• Which type of search do we use when we are searching for a solution to the 8-queens problem? Why?

- :

• Which type of search do we use when we are searching for a path between cities on a road network? Why?

- : Tree search, as we can backtrack to other paths based off given criteria.

3. What are the criteria used to compare search algorithms?

- There are four criteria comparing the various search tree algorithms:
  - 1. Completeness: Does it return a solution in finite time, if one exists?
  - 2. Optimality: Does it return the optimum solution?
  - 3. Time Complexity: How many nodes are generated until a solution is found?
  - 4. Space Complexity: How many nodes -maximum- we have to remember to find the solution?
- The following parameters are used to calculate and compare the performance of an algorithm:
- b: maximum branching factor
- d: the depth of the shallowest goal state
- m: maximum length of path in the tree

4. What are the properties of breadth-first (or depth-first, or uniform-first, or iterative-deepening depth-first, or bidirectional breadth-first) search? Justify the complexity bounds you report.

- Breadth-first:
  - Data Structure: First In First Out (FIFO) - Queue
  - Complete: if b is finite
  - Suboptimal: Optimal if all the costs of the edges are equal.
  - Time Complexity: $O(b^{d+1})$
  - Space Complexity: $O(b^{d+1})$
- Uniform-cost:
  - All nodes in fringe have associated path cost. The one with the minimum path cost is selected.
  - Parameters: C^* is optimal cost; ϵ is smallest edge cost ($\epsilon > 0$)
  - Complete and Optimal
  - Time Complexity $O(b^{(C^*/\epsilon)+1})$
  - Space Complexity $O(b^{(C^*/\epsilon)+1})$
- Depth-first:
  - Always expands leftmost child node fully before moving on to the next child node.
  - Data Structure: Last In First Out (LIFO) - Stack
  - Incomplete
  - Not optimal
  - Time Complexity $O(b^m)$ where m = maximum length across the tree
  - Space Complexity is O(bm) since we don't need to remember *all* children at any given point (only the leftmost)
- Iterative-deepening depth first search:

- Complete? Yes, because by iteratively increasing the limit, at some point the algorithm reaches a goal node and the algorithm never searches an infinite path.
- Time: $O(b^{(d)})$. Notice that the time complexity of IDDFS is better than BFS's. When the shallowest goal node is at level d, BFS will generate nodes at level d + 1. The number of nodes at level d + 1 is almost as many nodes on all the levels up to d. On the other hand, IDDFS is limited to generate nodes up to level d and consequently expands much less nodes than BFS.
- Space: O(bm). Same with the space complexity of DFS, since at each point in time IDDFS is executing a depth- first search and has only to remember the left siblings of the currently expanded node and of its ancestors. For example in Figure 11, when the node H is expanded at "Limit-3", the fringe includes also the nodes: I, E, C.
- Optimal? Yes, if all the edges have the same cost.

- Bidirectional search:
  - Runs two concurrent BFS searches up to level d/2. This isn't always possible because there can be infinite goal states, or the problem is simply not invertible.
  - Complete? Yes, in the BFS sense.
  - Time: $O(b^{d/2})$, because each BFS algorithm has to search only half the number of levels that the entire solution contains.
  - Space: $O(b^{d/2})$.
  - Optimal? Yes, if the edges have the same cost, as in BFS.

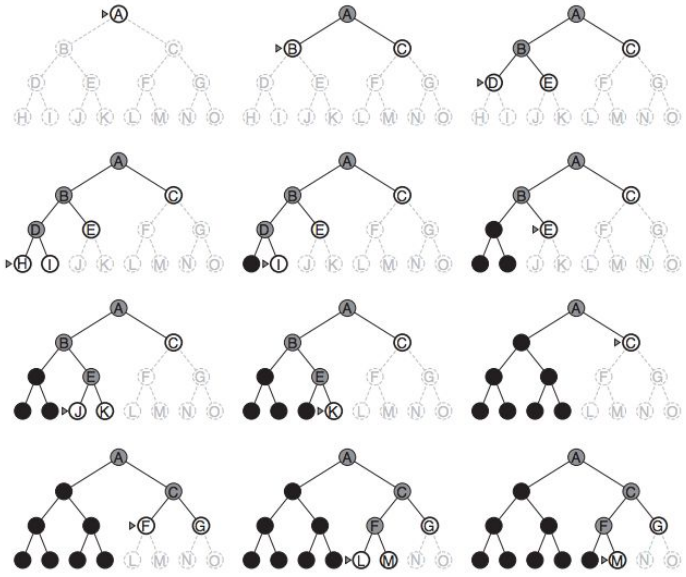


Figure 9: Depth-first Search.

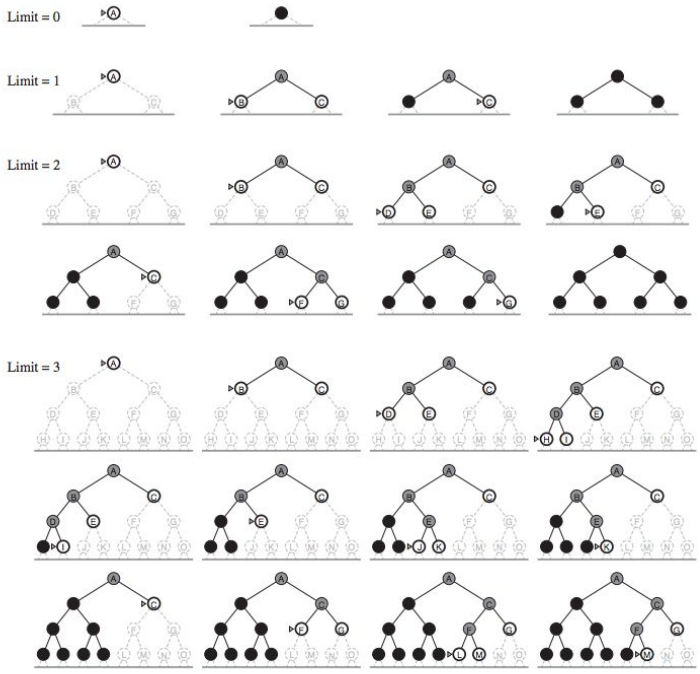
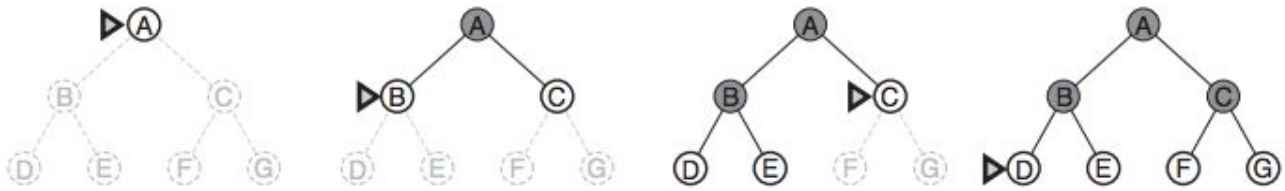


Figure 11: Iterative Deepening Depth-First Search

6.1 Breadth First Search



6.5 Bidirectional Search

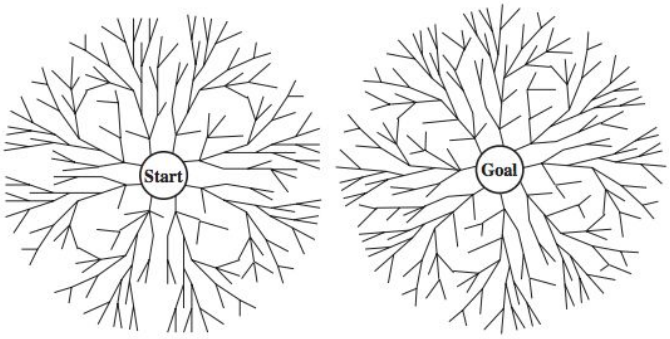


Figure 12: The graph of a bidirectional search tree.

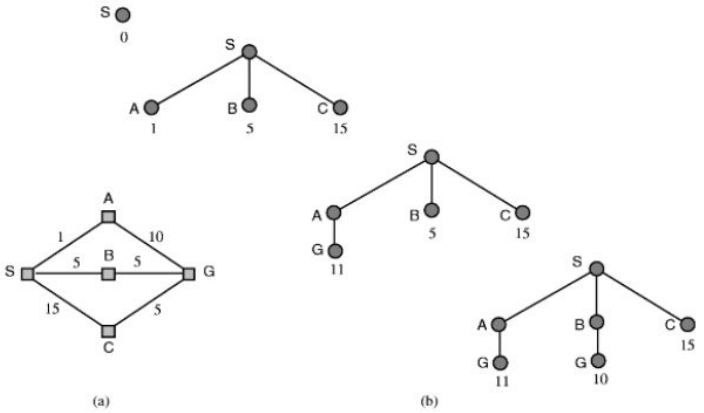


Figure 10: Uniform-First Search.

	DFS	BFS	US	IDDF	BDBF
Path:	Expands left most node all the way then goes back one and does right node	Expands each child before moving in depth	Expands each child, selects lowest cost clid to expand next	Basically BFS with a limit so that there are never infinite paths	Runs two BFS from each end to the middle
Complete:	NO	If b is finite	YES	YES	If b is finite
Optimal:	NO	If all cost of edge are equal	YES	If all the edges have the same cost	If all cost of edge are equal
Time:	$O(b^m)$	$O(b^{d+1})$	$O(b^{(C*/\epsilon)+1})$	$O(b^{(d)})$	$O(b^{d/2})$
Space:	O(bm)	$O(b^{d+1})$	$O(b^{(C*/\epsilon)+1})$	O(bm)	$O(b^{d/2})$
	B = max branching factor	D: depth of the shallowest path	M: max len of path	C* is optimal path	ϵ: smallest edge cost, > 0

5. Which uninformed search technique has the best time and space complexity? What is the time and space complexity?

- Best space complexity: iterative-deepening and DFS.
- Best time complexity: Bidirectional

6. Which algorithm would you use if you knew that the depth of the search tree is bounded and Why?

- Depth-First Iterative-Deepening. It has the same space complexity as DFS O(bd), has advantages of BFS (completeness) and is preferred for large state spaces where depth is known.

7. Prove that uniform-cost search and breadth-first search with constant edge costs are optimal when used in GRAPH-SEARCH.

- BFS is optimal when edges all have the same cost. Otherwise, not optimal but finds solution with shortest path length. Uniform Cost Search is exactly the same as BFS if all nodes have same costs.

8. Show a state space with varying step costs in which GRAPH-SEARCH using depth-first (or

iterative-deepening depth first) finds a suboptimal solution.

- :

9. Under which circumstances can you apply bidirectional search?

- If the problem is invertible, i.e the goal states can be measured.

10. You need an additional data structure in order to be able to solve GRAPH-SEARCH problems. How is this data structure called and how can you implement it?

- Closed list, which saves nodes that have already been expanded.

11. Using one or more of the criteria used to compare search algorithms, provide arguments to

support the following statements:

• Iterative deepening search is preferred over breadth-first search.

- The cost (time complexity) is better for IDDF then in is for BFS

• Bidirectional search is preferred over breadth-first search.

- This is true because the time and space complexity are both $O(b^{d/2})$, compared to $O(b^{d})$ in BFS.

2. Informed Search

1. What is the name of the overall search strategy that selects the fringe node which maximizes an evaluation function $f(n)$? Name two approaches that implement this search strategy? How do they define the evaluation function? What does a heuristic represent?

- The overall search strategy is called Heuristic/Informed search, or **Best First Search.** The two approaches that implement this are **Greedy** Best-First Search, where f(n) = h(n) and **A*** where f(n) = h(n) + g(n). Heuristic is distance from current node to goal node.

2. What are the properties of greedy best-first search? How does it work? What are the properties of A* in TREE-SEARCH and GRAPH-SEARCH? How does it work?

- Greedy Best-First Search
  - Space & Time Complexity: O(b^m)
  - Expands node closest to goal based on heuristic: f(n) = h(n)
  - Informed equivalent to DFS. Implemented using priority queue.
- A*
  - TREE-SEARCH
    - Allows repeated states
    - Optimal if h(n) is admissible
  - GRAPH SEARCH
    - Avoid repeated states
    - 2 solutions to retain optimality: once repeated state found, discard more expensive path, make sure the first path followed to a repeated state is optimal. If h(n) is consistent then A* is optimal

3. True or False?

• Greedy Best-First search with h(n) = 0 for all nodes n is guaranteed to find an optimal solution if all arc costs are 1.

- "Assuming all arc costs are 1, then Greedy Best-First search will find the left goal, which has a solution cost of 5, while the optimal solution is the path to the right goal, which has a cost of 3."
- No - GBFS will always expand the leftmost so if optimal is not leftmost, then no

• In a finite search space containing no goal state, the A* algorithm will always explore all states.

- TRUE : (according to google)

4. Prove the following statement: "If G2 is a node that corresponds to a goal state but a suboptimal path, A ∗ will select a reachable node n along the optimal path before selecting G2 for expansion"

- So first we see that h(g2) = 0 (because it's the goal node)
- And then we see that g(g2) > c* because g2 is a goal node that is connected to root but the suboptimal path (c*) is inside the path
- So f(n) <= C* which is < f(g2)
- So n would be selected first

5. What is the definition of an admissible heuristic? Prove that A∗ always finds the optimal path

in TREE-SEARCH if the heuristic is admissible.

- Admissible means the heuristic cost never over estimates the cost of the path

6. Is the "Manhattan distance" an admissible heuristic when planning on a grid?

- Yes

Is it true that the "Manhattan distance" is an admissible heuristic for finding the shortest path through the real Manhattan in New York City? [Hint: Go to Google maps and search for: "Broadway & W 33rd St, New York, 10001". Consider the distance between Madison Square Park and Times square.]

- Yes. the manhattan distance between TS and MSP is a straight line. The path that you would take from there is also (basically) a straight line. There is no way that manhattan would overestimate the cost (walking wise).

7. What is the definition of a consistent heuristic? Prove that a consistent heuristic is also

Admissible.

- Consistent heuristic: the cost from a location to neighboring vertex + heuristic from that vertex to the goal must always be less than or equal to the total heuristic from the location to the goal.
- The heuristic is always less than or equal to the estimated distance from any neighboring vertex (path cost) to the goal, plus the step cost of reaching that neighbor.
- If the heuristic is less than or equal to, then it is definitely less than (as what admissible is)

8. Show that if a heuristic h(n) is consistent, then the evaluation function used by the A* algorithm $f(n) = h(n) + g(n)$ is nondecreasing along any search path. Show that A* expands nodes with a non-decreasing order of evaluation function.

- : A consistent heuristic is h(n) <= c(N,P) + h(P) and h(G) = 0, where c is the step cost. Intuitively, this means that h(n) will always be less than (or equal to) the actual cost to get from G to N, and so as A* gets closer to G, the value of g(n) won't decrease since h(n) was already "admissible" (an underestimate).

9. Show that the first goal node expanded by the A* algorithm also corresponds to the optimal goal node. Show that A* expands all nodes with $f(n) < C*$, where C∗ the optimal solution cost.

- :

10. Under which circumstances is the A* algorithm complete?

- "A* is complete if the number of nodes for which f(n) < C∗ is finite."

11. Which optimal informed search algorithm expands fewer nodes than A* for the same heuris-

tic? Ignore the effects of ties.

- :

12. You might be provided with a graph and asked to show the sequence of nodes expanded by

A*, as well as the g, h and f values. You might be asked if the heuristic was admissible.

- :

13. Under which conditions will A* search expand the same nodes as Breadth-First search?

- :

14. Which of the following are admissible, given admissible heuristics h1, h2? Which of the following are consistent, given consistent heuristics h1, $h2$?

• h(n) = min{h1(n), h2(n)}

- Admissible means it will always be under the actual cost, if both h are under the actual cost, and you pick one of those then both will be admissible

• $h(n) = w * h_1(n) + (1-w) * h_2(n)$

- :

• $h(n) = m\{h1(n), h2(n)\}$

- Admissible means it will always be under the actual cost, if both h are under the actual cost, and you pick one of those then both will be admissible

15. What is the definition of the effective branching factor and how is it used to compare heuristics?

- If an algorithm has visited N nodes
  - solution was at depth d,
  - then b ∗ is the branching factor of the complete tree of length d with N nodes.
  - N + 1 = 1 + b ∗ + (b ∗) 2 + . . . + (b ∗) d
- Optimum and desired performance of b* is 1

16. Suggest possible directions for automatically designing heuristics.

- :

17. Give examples of relaxed versions of the 15-puzzle?

- Relaxed: optimal solutions to problems with fewer restrictions than the original definition
- Use subproblems:
  - Find the optimal solution for looking at just the first 8
  - Or the first 4 then the first 8 and finally the entire puzzle

18. What is the definition of a dominant heuristic?

- When h2(n) > h1(n), ∀n
- H2 dominates h1
- H2 has a lower effective branching factor

3. Adversarial Search

1. What is the definition of a two-player zero sum game?

- Deterministic
- Turn-taking
- two-player problems
- utility values of the two agents at the end of the game are always equal and opposite

2. What are the differences between a search tree used for adversarial search versus a tree for classical single-agent search problems?

- game tree: we need additional levels in order to express the actions of the opponent
- Piles: separate levels
- Terminal nodes: leaf nodes

3. Describe the operation of the minimax algorithm. What properties does it have in terms of time and space complexity?

- Minimax−Value(n) =
  - = Utility(n) if n = terminal state
  - Or $\max_{s \in succ(n)}$ MV(s) if n is max node
  - Or $\min_{s \in succ(n)}$ M V(s) if n is min node
- (depth first - search)
- Time complexity: O(b^m) b = # of moves at each iteration
- Space complexity: O(bm) m = max depth (max moves)

4. Why do we use depth-first search to solve adversarial search problems?

- You need to know the result of that particular move. So you need to go all the way down the tree to get that result. Wouldn't really make sense to do it using BFS

5. Describe the operation of alpha-beta pruning. What is the time complexity assuming perfect

ordering? What is the time complexity on average?

- Basically the same but we can avoid considering certain values
- Each node has an alpha and beta value [-inf, inf]
- It is set by the terminal node, and passed down to children
- Time (perfect pruning): O(b^(m/2))
- Time (average): O(b^(3m/4))

6. You might be provided a search tree that corresponds to a game and asked to identify which nodes will not be checked by the minimax algorithm if we apply alpha-beta pruning.

You might also be asked to compute the value at each node as computed by the minimax algorithm.

You might be asked if a different ordering of the terminal nodes would result in increased pruning.

- :

7. Are the following statements true or false?

• A player using minimax is guaranteed to play optimally against any opponent.

- False. Assumes the opponent is operating optimally.

• A player using alpha-beta pruning is guaranteed to play optimally against an optimal opponent.

- true dat

8. Do you know any other way to accelerate the performance of adversarial search other than

alpha-beta pruning?

- Using heuristics???
- :

9. What requirements should a heuristic function satisfy in heuristic adversarial search?

- Be fast to compute
- Order terminal states the same way the the utility function would
- correlate to the actual chances of winning for non-terminal states

10. Give examples of problems that might arise in heuristic adversarial search.

- Cutting off might lead to change the minimax value
- Some moves that are optimal, are hidden deep in the tree and cut off can lead to not accessing them

11. What are the differences between a search tree used for games with chance versus search

trees for deterministic games?

- Have an additional play “CHANCE”
- Find probability
- Scaling terminal node will affect tree
- Complexity of Expectiminimax: O(b^m * n^m), n is branching factor of change nodes
- Worse than minimax

12. Describe the operation of the expectiminimax algorithm.

- Expectiminimax(n) =
- U tility(n) n: terminal state
- maxs∈succ(n) EMMV (n) n: max
- P
- mins∈succ(n) EMMV (n) n: min
- s∈succ(n) P(s) ∗ EMMV (s) n: chance.
- Basically if you at state max/min you find the max/min
- Otherwise if you are at chance level, you take the prob of getting the leaf times the value and added those

13. What problem arises in heuristic adversarial search for games with chance in the definition

of the heuristic evaluation function that does not exist in deterministic games?

- Can’t order the same was at util function
- :

14. What are the difference between a search tree used for non zero-sum games with more than

2 players versus search trees for two-player, zero-sum games?

- A new ply is added
- Nodes store more than one value

4. Local Heuristic Search

1. When is it appropriate to use local search instead of classical search? What are the advan-

tages and the disadvantages of local search?

- When path not important
- Maximization problems
- Moves between neighboring nodes
- ADV:
  - Good space complexity
  - Easy to code
  - Solve problems w/out explicit notation of goal
- DADV:
  - Stuck in local maxima
  - Or in Plateaux: not exploring values that are the same

2. Say we have a search space that is very large with a very large branching factor at most nodes, there may be infinite paths in the search space, and we have no heuristic function. What search method would be good to use in this situation and why?

-

3. Describe the basic operation of a hill-climbing approach.

- You're at a node, if there is a neighboring node that is better value, the current node will become this node

4. How can you avoid the issue of plateaux in hill-climbing?

- Random walks - randomly pick a location that has equal fitness

5. What does "probabilistic completeness" mean and how can you achieve a hill-climbing ap-

proach that is probabilistically complete?

- As time passes, probability of being complete → 1
- Restaring?
- :

6. You might be provided with a toy-problem (e.g., the n-queens problem) and asked to define a heuristic evaluation function that will allow you to apply hill-climbing.

- :

Given your heuristic function, you might be asked to solve a specific instance of such a toy problem with hill- climbing.

- :

7. Describe the operation of simulated annealing. What is the motivation behind the development of this local search technique? How does the algorithm behave at very high "tem- peratures", and how it behaves at very low "temperatures"? When is simulated annealing probabilistically complete?

- Hill climbing with random walks
- ΔE = value(current) − value(neighbor)
- T = which decreases by some function at the beginning of every iteration. How fast or slow the temperature decreases is defined by the so called "annealing" or "cooling schedule"
- "The probability decreases exponentially with the badness of the move"
  - With high temperatures: Increased probability of going down hill
  - With low temperatures: Primarily hill-climbing
- Probabilistically complete: Finds optimal solution with a probability → 1 (assuming temperature decreases slowly enough)
- Space complexity: O(1)
- Complete, ΔE: e^($-\Delta E/T$)

8. For what types of problems will hill climbing work better than simulated annealing? In other words, when is the random part of simulated annealing not necessary? Is Simulated Annealing with a constant, positive temperature at all times the same as Hill-Climbing?

- If the problem is specific problems?

9. What is the main principle behind local beam search? How is it different from traditional hill

climbing? What are the properties of local beam search?

- Traditional local search remembers and evolves a single state.
- Local Beam Search: remembering multiple states simultaneously
    - Advantage:
    - Space complexity is O(n), where “n” is the number of individuals in the population
    - Disadvantage:
    - Not complete and not optimal.
    - Can get stuck in local minima.
    - Can easily degrade to simple hill climbing.

10. What are the main principles behind genetic algorithms? How are genetic algorithms different from local beam search? What are the properties of genetic algorithms?

- an individual is not a direct decedent of a single individual from the previous
- population.
- Have a selection, reproduction, and mutation processes
- Advantages:
    - Effective on very large search spaces.
    - Effective on a wide range of problems
- Disadvantages:
    - Not necessarily efficient or complete

11. You might be asked to design a solution based on genetic algorithms for a specific problem

Components:

Initial Population - starting group

Fitness Function - determines survival

Selection - parents chosen

Crossover - recombining elements

Mutation - small errors in copying

12. Consider a genetic algorithm using only selection and mutation operators, that is without crossover. Is this equivalent to any other search algorithm? Justify your answer.

- This is basically like random walkin, cause mutation will give you a random number just like random walking, you pick a random number.
    - ???? help

	Genetic	Local beam search	Simulated annealing	Hill climbing
path:	selection, reproduction, and mutation processes	Hill climbing with more than one state	Hill climbing with random walks “ instead of picking best move, it picks random move. If the move actually improves the situation, it always proceeds”	You’re at a node, if there is a neighboring node that is better value, the current node will become this node
advantages :	Effective on very large search spaces. Effective on a wide range of problems			
efficient :	Not really			
Complete:	Not really			
Optimal:				
Time:		ΔE: $e^{(-\Delta E/T)}$		

space:				

5. Constraint Satisfaction Problems (CSPs)

1. What is the definition of a constraint satisfaction problem?

- Problems which conform to a standard, structured and very simple representation
- Have variables and constraints
- Consistent: variables are not violating the constraints
- Complete: every variable is used

2. What are two ways to approach the solution of constraint satisfaction problems? What techniques are more appropriate for each approach?

- Incremental formulation
  - Noting filled in -> fill one within the constraints, -> 1 cost for every step -> goal: assignment is complete
- Complete-state formulation
  - Everything fill (no constraints) -> change one thing to fit constraints-> no path cost (DNA) -> goal: assignment is consistent

3. You might be provided a specific constraint satisfaction problem and asked to devise an incremental or local-search approach for solving it.

- :

4. Why do we use a variation of depth-first search to solve constraint-satisfaction problems? How is this variation called?

- Has best space complexity
- Once depth is reached, its complete
- Called Backtracking
- algorithm does not always have to reach the leaf nodes of the tre
  - soon as it discovers that it has reached an inconsistent assignment
    - stop searching down this branch
    - backtracks to the parent node
    - evaluate alternative assignments

5. What heuristics do you know for ordering variables in backtracking search? What heuristic would you use to order value assignments in backtracking search?

- Minimum Remaining Values (MRV) heuristic
  - specifies an order for choosing which variable to assign next
  - Variables that are most constrained have higher priority
- Degree heuristic
  - select the variable that is involved in the highest number of constraints
  - Tie breaker for MRV
- Least Constraining Value heuristic
  - prefers the value that rules out the fewest choices for the neighboring variables

6. Describe the forward checking heuristic.

- when variable X is assigned to go to variable Y , where Y is a neighbor of X in the constraint graph and delete all the possible values for Y that are inconsistent with X's choice.
- For ex. If you color WA red, then all the negibohr's domain will delete red

7. Describe the arc consistency heuristic.

- An approach that detects early on inconsistencies
- Given the domain of neighboring variables X and Y , the arc between the two is consistent if ∀ values x of X, ∃ some value y of Y that is consistent with x.

8. What is the difference between the forward checking heuristic and the arc consistency heuris-tic? In other words, when will each heuristic detect an inconsistent assignment?

- :

9. What is the idea behind intelligent backtracking? How does backjumping operate?

- attempts to detect the variable that caused the inconsistency and the one variable that the backtracking algorithm should backtrack to

10. You might be provided an example of a constraint satisfaction problem (e.g., a map coloring problem) and asked to predict the effects of certain heuristics in backtracking search. (e.g., if we assign this variable a certain value, what will be the result of Forward Checking, etc.)

- :

6. Logic-based Inference and Satisfiability

1. What is the conjunctive normal form (CNF)? Given an example of a logical sentence in CNF.

- This means that any complete search algorithm, applying only the resolution rule, can derive any conclusion entailed by any knowledge base in propositional logic
- corresponds to a conjunction of clauses
- Has to be in form $(l_{1,1} \vee \ldots \vee l_{1,k1}) \wedge (l_{2,2} \vee \ldots \vee l_{2,k2}) \wedge \ldots$
- So alpha $\Leftrightarrow$ beta turns into (not alp or beta) and (not beta or alpha)

2. Use logical equivalences to show that the statement $\alpha \Leftrightarrow \beta$ (or a different logical statement) can be turned into conjunctive normal form.

- 1. Apply biconditional elimination : (alp => beta) and (beta => alp)
- 2. Apply implication elimination: **((not alp or beta) and (not beta or alpha))**

3. Show that the clause $(\neg P_1 \vee \ldots \neg \vee P_m \vee Q)$ is logically equivalent to the implication sentence $(P_1 \wedge \ldots \wedge P_m) \Rightarrow Q$.

-

4. What is the resolution rule and why is it important for logical inference?

- specifies that if a variable appears in two disjunctive expressions
  - with opposite signs
- new disjunctive expression is implied where all the variables from the original disjunctions appear except from the one that appears with opposite signs
- Basically $\Rightarrow$ variable that appears with both the positive and negative sign will be removed

5. Describe the steps of the satisfiability algorithm?

- Given KB (a set of rules / constraints) (knowledge base)
- prove α is True
- Represent KB in Propositional Logic
  - 2. Take sentence $(KB \wedge \neg\alpha)$ and turn it into CNF.
  - 3. Apply repetitively the resolution rule until 1 of the following 2 happens:
    - a) There is no new clause you can create with resolution. (KB does not entail α)
    - b) 2 clauses resolve to the empty clause. $(KB \wedge \neg\alpha)$ is unsatisfiable, so $(KB \models \alpha)$

6. You might be provided with a knowledge base KB and asked to use the satisfiability algorithm in order to prove that it entails a new sentence α: $KB \models \alpha$.

- :

7. What is the Davis-Putnam algorithm? What strategy does it follow to solve logic-based inference problems?

- Models constraint satisfaction problems
- using propositional logic and solves them
- through a backtracking depth-first search

8. Describe heuristics for the Davis-Putnam algorithm.

- Early Termination Heuristic
  - conjunction of two or more clauses must be true or false
  - clause is true if any literal is true
  - (A ∨ B) ∧ (A ∨ C) -------> evaluates true if A is true, regardless of the values of B and C
- Pure Symbol Heuristic
  - symbol that always appears with the same ”sign” in all clauses
  - (A ∨ ¬B),(¬B ∨ ¬C), and (C ∨ A), ---->
    - A is pure
    - B is pure
    - C is IMpure
- Unit Clause Heuristic
  - clause with a single literal
  - or a clause in which all literals but one are assigned false
  - given B = f alse, and a clause (B ∨ ¬C),
    - clause becomes a unit clause
    - it is equivalent to (f alse ∨ ¬C)

9. What are three alternative methodologies to logic-based inference problems.

- WALKSAT Algorithm
  - hill climbing technique specifically designed for the satisfiability problem
  - greedily (minimize the number of unsatisfied clauses)
  - • randomly
  - Is WALKSAT complete?
  - • If it returns a model, it has found a solution
  - • If it hasn’t, we do not know
- First Order Logic
  - ∀ (always), ∃ (exists) • Functions • Equality
- Second- Order Logic
  - Always True • Until, After then

7. Classical Planning

1. Describe the version of the Planning Domain Definition Language (PDDL) that was covered in class. Given examples of challenges that can be described by PDDL.

- States are represented as conjuncts of positive literals. (everything not mentioned is assumed to be false) Actions are specified by giving their preconditions and their effects.
- Ex. Air cargo transportation, Hanoi tower/stacked blocks

2. You may be provided a problem and asked to build its PDDL specification.

- init(at(c1, SFO) ^ at(c2, JFK) ^ at(p1,SFO) ^ at(p1, JFK) ^ cargo(c1) ^ cargo(c2) ^ plane(p1) ^ plane(p2) ^ airport(SFO) ^ airport(JFK))
- goal(at(c1,JFK)^at(c2,SFO))
- action(load(c, p, a),
  - PRECOND: at(c, a)^ at(p,a)^cargo(c)^plane(p)^airport(p)
  - EFFECT:-at(c, a)^ in(c,p))
- action(unload(c, p, a),
  - PRECOND: in(c, p)^ at(p,a)^cargo(c)^plane(p)^airport(p)
  - EFFECT:at(c, a)^ -in(c,p))
- action(fly(p, from, to),
  - PRECOND: at(p, from)^ plane(p)^airport(from)^airport(to)
  - EFFECT:-at(p, from)^ at(p, to))

3. What is the idea in forward state search for classical planning?

- Start from the initial state and define actions that are applicable to generate children states. Successors are generated by adding positive effects and deleting negative ones

What are the challenges of forward state search?

- Number of actions can be larger than backwards state-space search. All actions, even irrelevant actions are considered. Quickly gets bogged down without a good heuristic

4. What is the idea in backward state-space search for classical planning?

- Start from the goal state and consider actions whose effects agree with the state specification.

What are the challenges in backward state-space search?

- Can be difficult to implement if the goal state is described by a set of constraints rather than explicitly stated. Also defining predecessors is not always obvious.

5. How can you define heuristics for classical planning?

- **for relaxed version
- -ignore precond
- -consider only positive effects -> count # of actions that give goal from current state.

8. Intro to Decision Theoretic Agents and Probability Theory

1. What is the focus of decision theory? What are the basic steps of a decision theoretic agent?

- DecisionTheory = ProbabilityTheory + UtilityTheory
- DT agents: act intelligent under uncertainty
- Steps:
  - Update a belief-state distribution based on actions/percepts
    - (answering the question: what is the most probable state the world is currently in?)
  - Calculate outcome of actions given action descriptions + current belief state
    - (compute the utility of each action)
  - Select action with the highest expected utility given probabilities of outcomes and utility information
  - Return action

2. You might be asked to provide any of the rules of probability theory discussed in the class:

• Kolmogorov's axioms,

- • 0 <= P(a) <= 1, ∀a • P(true) = 1, P(f alse) = 0 • P(a ∨ b) = P(a) + P(b) − P(a ∧ b)

Axiom	Meaning
0 <= p(a) <= 1, ∀a	The prob of a is in between 0 and 1
P(true) = 1, P(false) = 0	If a is true then the prob of a = 1. If a is false, the prob of a is 0
P(a ∨ b) = P(a) + P(b) − P(a ∧ b)	prob(a or b) = prob(a) + prob(b) minus the inclusion of a AND b

-

• Product rule - with and w/o normalization,

- p(a|b) = p(a ^ b) / p(b) ⇒ prob of a given b = prob a AND b divided by prob of b

• Bayes rule - with and w/o normalization,

- p(a|b) = [p(b|a) * p(a)]/p(b) ⇒ p(a given b) = p(b given a) times p(a) all over p(b)
- p(y|xe) = [p(x|ye) * p(y|e)]/p(x|e) ⇒ here, X,Y are events and e is a background evidence
- With normalization:
- < P(b|a), P(¬b|a) >=< αP(a|b)P(b), αP(a|¬b)P(¬b) > ⇒ when comparing p(b given a) and p(NOT b given a) use this
  - p(m|s) = α < P(s|m) × P(m), (P(s|¬m) } but we have to know this variable)×P(¬m) >

• marginalization and conditioning,

- p(x) = sum x p(y,x) - marginalization
- Conditioning
  - p(x) = sum y p(x|y)*p(y) -or
  - p(x) = sum y p(x,y|z)*p(y|z)

• independence and conditional independence properties.

- p(a|b) = p(a) ⇒ if a variable is INDEPENDENT, then p(b given a) = p(a)
- p(b|a) = p(b)

- p(a^b) = p(a)*p(b) ⇒ if a variable is INDEPENDENT, then p(a AND b) = p(a) times p(b)
- If you draw a venn diagram you see that the circles of A and B never touch so these following rules can apply

3. You might be asked to compare the value of P(a|b) and P(¬ a|b) in a way that minimizes the number of probabilities that we have to be aware of.

- Isn't this just bayes rule with normalization????

4. Assume two random binary variables α,β. What is the minimum number of distinct probabilities that you need to know to compare P(α|β) and P(¬α|β) using the Bayes rule? [Clarification of "distinct": if you know P(γ) then you also know P(¬γ) = 1 − P(γ).]

- P(α), P(β), P(β|α)

5. What does the Naive Bayesian model specify?

- Naive bayes thinks all variables are independent of each other

9. Bayesian Networks: Properties and Exact Inference

1. What is the definition of a Bayesian network?

- A graphical representation that expresses conditional independence properties among variables of a problem involving uncertainty.
- -nodes are random variables
- -directed links connect pairs of nodes x->y (x is parent of y) when x has direct influence over y
- -conditional probabilities p(y|parent{y})

2. Why can we answer any query in a probabilistic domain given the Bayesian network that

involves all the variables in this domain and their independence properties?

- We can answer any query due to the joint probability distribution information we gain that is available on the Bayesian network.

3. How can we compute the joint probability distribution P(X1,...,*Xn*) from the conditional probabilities P(X |Parents(X)) stored on the Bayesian network that involves variables X1,...,*Xn*?

- Product rule
- $$\begin{aligned} P(X_1,\ldots,X_n) &= P(X_n|X_{n-1},\ldots,X_1)\cdot P(X_{n-1},\ldots,X_1) \\ &= P(X_n|X_{n-1},\ldots,X_1)\cdot P(X_{n-1}|X_{n-2},\ldots,X_1)\cdot\ldots\cdot P(X_2|X_1)\cdot P(X_1) \\ &= \textstyle\prod_{i=1}^{n} P(X_i|X_{i-1},\ldots,X_1) \end{aligned}$$
- ==> $$P(X_1,\ldots,X_n) = \prod_{i=1}^{n} P(X_i|Parents(X_i))$$

4. You may be provided with a Bayesian network and asked to compute the joint probability distribution. You can also then be asked to compute conditional probabilities that involve the variables of the Bayesian network. The conditional probability might:

• Involve all the variables in the Bayesian network

• Involve a subset of the variables in the Bayesian network

- (P(a, b) = P(b|a)P(a))
- Consider, for example, a problem with n = 30 nodes, each with five parents (k = 5). Then the Bayesian network needs n · 2 k probabilities to answer all queries (that is 960 numbers for our example), instead of 2n probabilities (=1 billion numbers) that we have to specify for the joint probability distribution.

5. How can we compute exactly a conditional probability of the form P(A|B) (without a normalization factor) from full joint probability distributions of the form: P(A,B) and P(¬A,B).

-

6. How does exact inference by enumeration work in Bayesian networks? You can be given a Bayesian network (e.g., such as the Burglary-Alarm problem) and asked to compute a conditional probability by applying exact inference.

- the complete set of variables involved in a problem is: X ∪E∪Y , and we want to compute a probability of the form: P(X|e)
- For example, what is the probability that a burglary occurred if both John and Mary called?
    - ==> P(burglary|JohnCalls = true, MaryCalls = true)?

$$
\begin{aligned}
P(b|j,m) &= \alpha \cdot P(b,j,m) && \text{(product rule)}\\
&= \alpha \sum_e \sum_a P(b,e,a,j,m) && \text{(conditioning)}\\
&= \alpha \sum_e \sum_a P(b)P(e)P(a|b,e)P(j|a)P(m|a) && \text{(Bayesian network)}\\
&= \alpha P(b) \sum_e P(e) \sum_a P(a|b,e)P(j|a)P(m|a) && \text{(moving terms out of the summations)}
\end{aligned}
$$

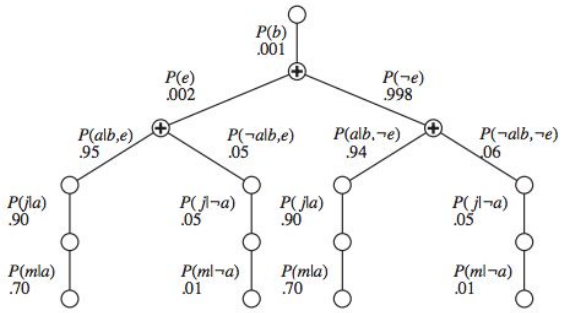


Figure 3: The enumeration tree for computing the probability $P(b|j,m)$.

7. What are the basic ideas behind Variable Elimination? How is it advantageous over Inference by enumeration? You might be provided a Bayesian network and asked to compute a conditional probability by applying Variable Elimination.

10. Approximate Inference in Bayesian networks

1. Which techniques do you know for approximate inference in Bayesian networks? What is the basic idea / methodology behind approximate methods for inference in Bayesian networks?

- 1. Direct Sampling
- 2. Markov Chain Simulation
- employ sampling in order to deal with the computational problems that arise in Bayesian networks

2. How does direct sampling work? You might be provided with a Bayesian network and a series of "random" numbers and asked to sample an atomic event by employing direct sampling.

- when you have a probability it is possible to sample from it
- we sample each variable in topological order according to the conditional probability over its parents
-
- That is a good explanation of finding an atomic event

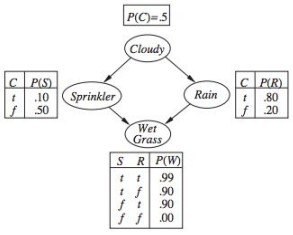


Figure 1: The Bayesian network for the rain-grass example.

- We first sample the value of the variable "Cloudy". $P(Cloudy) =< 0.5, 0.5 >$. Assume that sampling returns: "Cloudy"= true.
- We then sample the value of the variable "Sprinkler". Since "Cloudy" is true, we have to use the corresponding conditional probability: $P(Sprinkler|Cloudy = True) =< 0.1, 0.9 >$. Assume that the sample returns: "Sprinkler"=false.
- We then sample the value of the variable "Rain". Since "Cloudy" is true, we have to use the corresponding conditional probability: $P(Rain|Cloudy = True) =< 0.8, 0.2 >$. Suppose that the sampling process return: "Rain"=true.
- We then sample the value of the variable "Wet Grass". Since "Sprinkler" is false and "Rain" is true, we have to use the corresponding conditional probability: $P(WetGrass|Sprinkler = False, Rain = true) =< 0.9, 0.1 >$. Assume that the sampling returns true.

3. What direct sampling algorithms do you know to compute conditional probabilities? What are the basic ideas behind them? How do they work? You might also be provided a network and "random" numbers and asked to imitate the operation of these algorithms?

- 1. Rejection Sampling
  - we will produce the samples with the same approach as we described above with direct sampling
  - we will reject those samples that do not match the evidence
  - rejects too many samples
- 2. Likelihood Weighting
  - fix the evidence variables E and sample only the remaining variables X and Y

    Consider, for example, that we want to sample atomic events with likelihood weighting so as to compute the conditional probability: $P(Rain|Sprinkler = true, WetGrass = true)$. We show here the generation of a specific sample:

    - Initially the weight of the sample is set to 1: $w = 1$.
    - Then we sample the value of the random variable "Cloudy". $P(Cloudy) =< 0.5, 0.5 >$. Assume that sampling returns: "Cloudy"= true.
    - "Sprinkler" is an evidence variable with value true. For the evidence variables we have to update the weight:

      $$w \leftarrow w \times P(Sprinkler = true|Cloudy = true) = 0.1$$

    - Then we sample the value of "Rain" according to the appropriate distribution: $P(Rain|Cloudy = True) =< 0.8, 0.2 >$. Assume that it returns true.
    - "WetGrass" is an evidence variable with value true and we update the weight:

      $$w \leftarrow w \times P(WetGrass = true|Sprinkler = true, Rain = true) = 0.099$$

  - The question that arises is whether this process generates a consistent estimate??
    - Consistent: it has to conform to the limit to true distribution
  - This approach becomes problematic as the number of evidence variables increases and we are sampling from an increasingly smaller subset of the full joint probability distribution

4. What is the Markov Blanket of a random variable in a Bayesian network? Why is it important?

- We change the previous sample
- Sample a value for one of the non-evidence variables Xi.
- Take in neighbouring values into account before sampling Xi

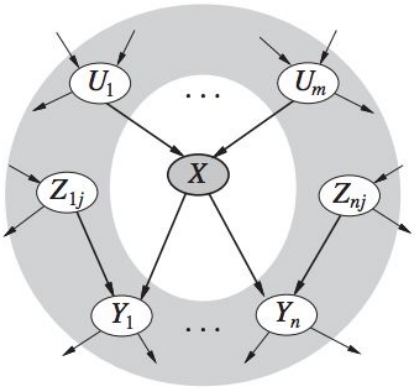


Figure 2: The Markov Blanket of variable X.

5. How does the Markov Chain Monte Carlo (MCMC) algorithm work? Imitate the algorithm's operation given a Bayesian network a series of "random" numbers.

- First we pick a non evidence variable and find the atomic event from there. Pick another one and so on.
  1. At each iteration we can sample the value of non-evidence variables, consequently we will sample either "Rain" or "Cloudy". Let's say we pick "Cloudy" and then we sample it, given its Markov blanket ("Sprinkler" and "Rain"). Assume that the sampling process returns: false. Then we have created a new atomic event $< false, true, false, true >$.
  2. Let's say that during the second iteration we pick "Rain". Then we sample this variable according to its Markov blanket (all the remaining variables). Assume that the result of the sampling process is: "Rain"=true. Then the new state that we have produced is: $< false, true, true, true >$.
- Then we add up the prob:

  In order to compute now the probability $P(Rain|Sprinkler = true, WetGrass = true)$, we just count the number of atomic events that have "Rain"=true. The important property of the MCMC approach is the following:

  The sampling process settles into a "dynamic equilibrium", where the fraction of time spent in each state is proportional to its posterior probability.