CHEATSHEET

SEARCHES:

Statements

① infinite loop, finite graph
+ve edge weights ⇒ no soluⁿ

② infinite loop finite graph
+ve edge weights ⇒ a soluⁿ exist

③ infinite loop finite graph
+ve edge weights, WHEN soluⁿ exits

④ Gaurranteed to find optimal solⁿ when all edge weight = 1 + solⁿ exists.

⑤ Gaurranteed to find optimal solⁿ when all edge weights +ve & solⁿ exists.

NO	DFS	BFS	UCS	A*	DFS	BFS	UCS
①	✓	✓	✓	✓			
②	✓						
③	✓				✓		
④		✓	✓	✓		✓	✓
⑤			✓	✓			✓

ADMISSIBLE:
$\forall n, 0 \le h(n) \le h^*(n)$

CONSISTANT:
$\forall a, c$
$h(A) - h(C) \le \text{cost}(A, C)$

UNIFORM SEARCHES

Criteria	BFS	UCS	DFS
Complete	✓	✓	✗
Time	$O(b^d)$	$O b^{c^*/\varepsilon}$	$O(b^m)$
Space	$O(b^d)$	$O b^{c^*/\varepsilon}$	$O(bm)$
Optimal	✓	✓	✗

INFORMED SEARCH

manhattan $= |x_1 - x_2| + |y_1 - y_2|$

greedy : not complete not optimal
fast + forward cost w heuristic

A* : complete, optimal, fast
total cost w proper heuristic

if h → admissible
use A* tree search → optimal solⁿ
if h → consistant
use A* graph search → optimal solⁿ

incosistant heuristic /inadmissable : greedy, UCS

consistant heuristic admissable : A*

if a solⁿ uses a good heuristic, C
all other admissible heuristic = C
consistancy ⇒ admissibility

CSP :

- how many possible assignments?
$O(d^N)$ n-vars d-domain size

unary constraint : involve a single var
binary constraint: involves two var

Advantages:
1) Memory
2) reasonable solns to large state space
3) good for optimization.

FORWARD CHECKING:

when a value assigned to a variable x, cross out values that share the same same constraint as x.

ARC CONSISTANCY:

An arc $X \rightarrow Y$ is consistant iff for EVERY x in the tail there is some y that can be assigned WITHOUT violating a constraint.

$O(n^2 d^3)$ can be reduced to $O(n^2 d^2)$

MRV	LRV
choose vals w fewest valid remaining values.	select vals that prunes least values from domains of unassigned vals.

LOCAL SEARCH:

① random assignment
② iteratively select random var + reassign its value to one the violates fewest constraints
③ do until no more constraint violations.

BRANCHING FACTOR:

1st : root : nd   each step
2nd : $(n-1)d$ 1 less val

general math:
$d^n n(n-1)(n-2) = d^n n!$

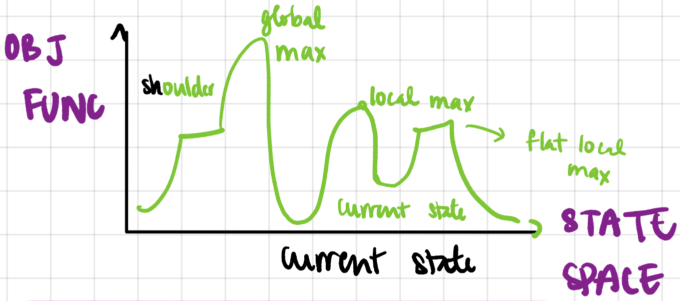
TREE STRUCTURE:

Time: $O(nd^2) \rightarrow O(d^N)$

CUTSET CONDITIONING:

find smallest subset of vars st that removal → trees.
Time: $O(d^c (n-c) d^2)$

HILL CLIMBING:



SIMULATED ANNEALING

escape local maxima by allowing some bad moves. but gradually ↓ size + frequency

energy ↑ @ next state go to state

energy ↓ @ next state dont go to state

$\boxed{T \propto \text{randomness}}$

LOCAL BEAM SEARCH:

Time step # 1 pick 2 states

Time step # 2 generate children " "

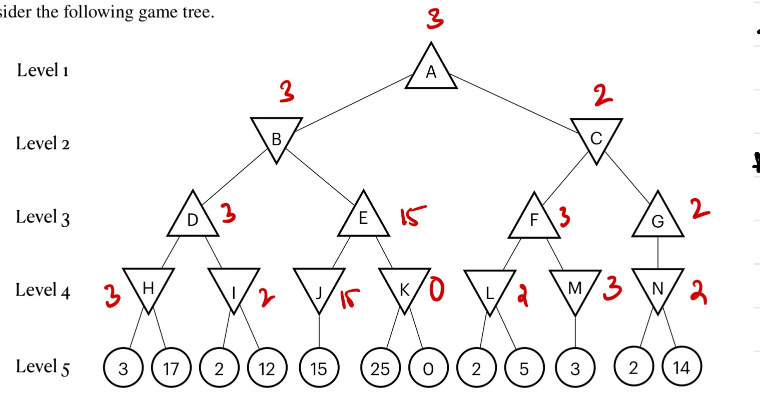
Time step # 3 " " until we find soln we are happy with.

MINMAX:

Time complexity: $O(b^m)$

similar to DFS, post order traversal → all subtrees before root. maximizer minimizer.

Consider the following game tree.



TRUTH TABLES:

Demorgan:

$\neg(a \vee b) : \neg a \wedge \neg b$

$\neg(a \wedge b) : \neg a \vee \neg b$

$\neg P \vee Q \quad : (P \to Q) \wedge (Q \to P)$

$(P \wedge Q) \vee (\neg P \wedge \neg Q)$

$A \models B$ iff $A \Rightarrow B$

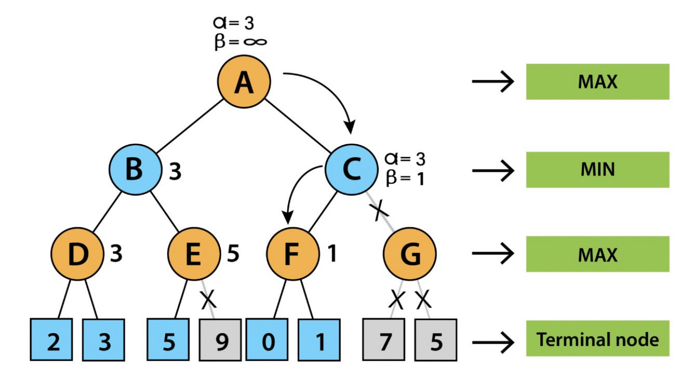
$A \not\models B \qquad A \wedge \neg B$ not valid.

satisfiable: exist 1 T

valid : True in all cols

ALPHA-BETA PRUNING:

Time complexity: $O(b^{m/2})$



stop looking as soon as n's val can at least = optimal val of n's parent.

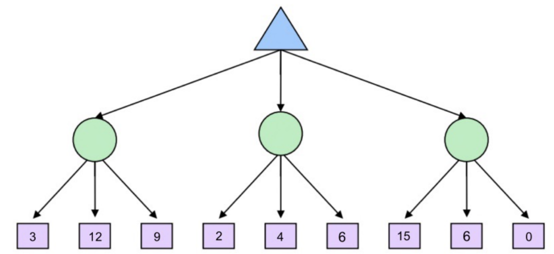
* 1st group of terminal nodes cannot be pruned.
* suboptimal when we prune low bound but optimal on upper bound.

Zero-sum:

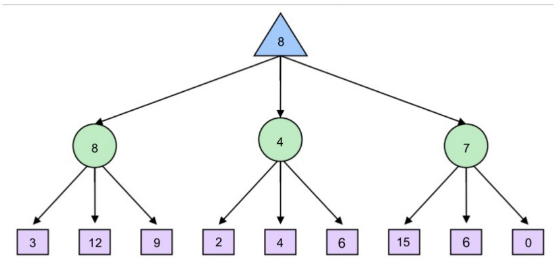
⇒ opponent's utility is the -ve of your utility

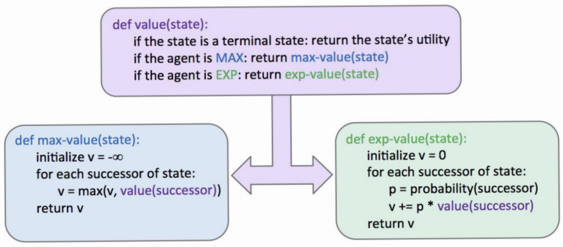
EXPECTIMAX:

Before we continue, let's quickly step through a simple example. Consider the following expectimax tree, where chance nodes are represented by circular nodes instead of the upward/downward facing triangles for maximizers/minimizers.



Assume for simplicity that all children of each chance node have a probability of occurrence of $\frac{1}{3}$. Hence, from our expectimax rule for value determination, we see that from left to right the 3 chance nodes take on values of $\frac{1}{3}\cdot 3+\frac{1}{3}\cdot 12+\frac{1}{3}\cdot 9=\boxed{8}$, $\frac{1}{3}\cdot 2+\frac{1}{3}\cdot 4+\frac{1}{3}\cdot 6=\boxed{4}$, and $\frac{1}{3}\cdot 15+\frac{1}{3}\cdot 6+\frac{1}{3}\cdot 0=\boxed{7}$. The maximizer selects the maximimum of these three values, $\boxed{8}$, yielding a filled-out game tree as follows:





∀ agent controlled states $V(s) = \max_{s' \in successors(s)} V(s')$

∀ chance states $V(s) = \sum_{s' \in successors} p(s'|s)V(s')$

∀ terminal states $V(s) =$ known.

MONTE CARLO TREE SEARCH:

→ large branching factor, minmax can't be used anymore.

- Eval by rollouts: from s play many times count wins/losses.
- selective search: explore parts of tree w/o constraints.

UCB1(n)

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

total no of wins for parent(n)

no of rollouts from n