CSCD84 ①

AI - planning dynamic problems (adapting)

Applications of AI

Robotics → Planning
- → Control
- → Mapping

• Language Processing
- → Translation → structure, semantics, distributions
- → Question Parsing — Wolfram Alpha
- → Search by similarity Google

Games → Competitive playing against AI chess

Logic → Answering Questions
- → Theorem Proving
- → Automatic Software Verification (formal meth
- → Deduction
- → Constraint satisfaction

Decision Making → Expert Systems

Classification → spam / ham
- → event recognition
- → recognition

Agent ← Intelligent
- → An entity
- → sense environment
- → carry out actions
- → achieve given goal

• Reactive (Reflex)
• Simulate / Predict

Utility function ()
maximize utility

reactive

↑

Cat & Mouse < see a cat run away
Smell choose - follow

Simulate & predict → set of possible actions
utility function.

- look at maze choose safer path.

model <u>of environment and other agents</u>

SEARCH

Looking for optimal actions
promising/good

Scheduling → try all possible solutions.

Components of Search problems
valid agent actions
* State Space → All $\boxed{\text{possible}}$ configurations of the environment & any agents
eg: All possible places cats & mice can be.

* function that determines goal

How to represent S.S
GRAPHS

loops are possible.

$\boxed{\text{Tree}}$ → you don't miss any good (optimal) paths soluti

CSCD84 ②

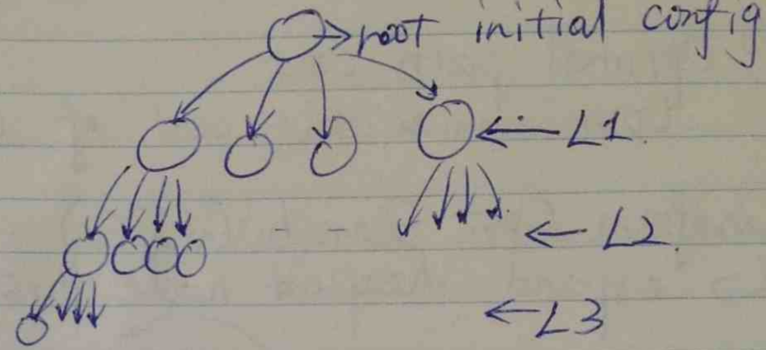
Successor function

- for any config, determines possible succeeding configs.

$\boxed{\text{Don't forget the path.}}$

model -> agents

Search Tree



Search
BFS. DFS.

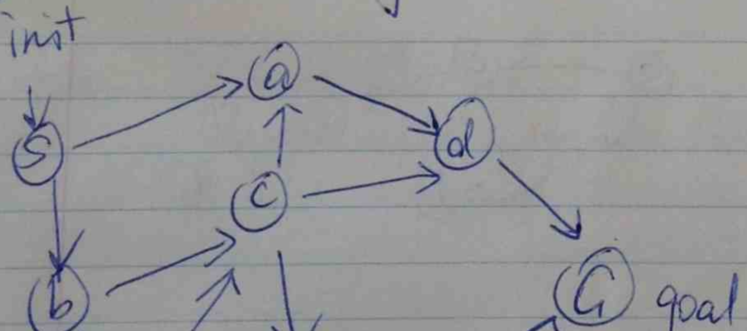
Overview of general search alg -> state/configuration possible sol'n.

while unexpanded nodes remain

* Policy DFS BFS <- select node to expand

↳ check if it is a goal

↳ if not, add unexpanded children to list of nodes to search.



get a goal as fast as possible, get a better goal

- multiple goal states
- some goals can be better than others.

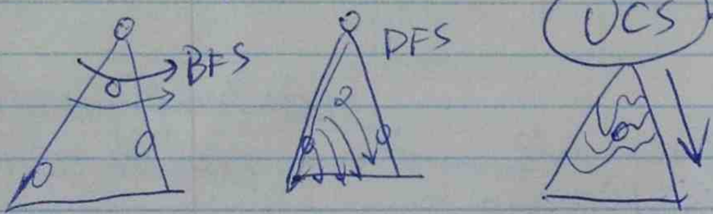
→ what does it mean better goal?
- configuration/state is better
- ~~often~~ cost of path

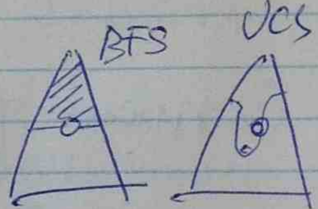
BFS/DFS don't care about the cost of actions

? Optimal path cost
↳ minimize the cost of agent's actions

Uniform Cost Search (UCS)
↳ expand cheapest node first

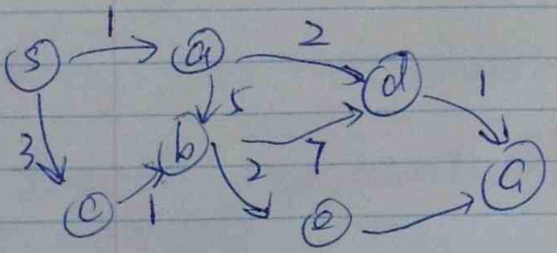
UCS → Dijkstra's method

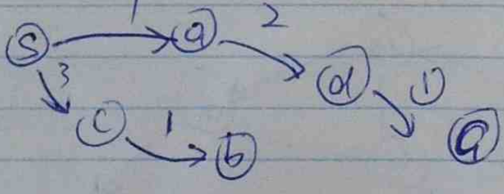




In general, ~~USC~~ UCS finds it faster.

UCS → cost distance from (S)
cost from (S)





Heuristic Search
↳ Attempt to measure how "close" a state is to a goal

cost of actions needed to get to

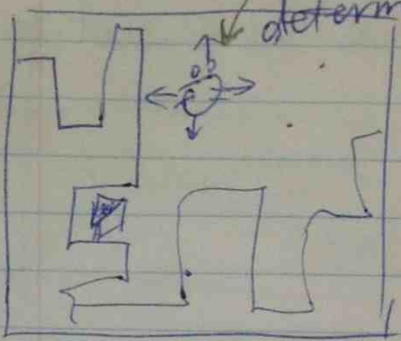
→ heuristic search cost

"cost" ← value of a node → $f(n) = g(n) + h(n)$ → heuristic cost of getting a solution

$g(n)$: cost of getting to node from S

CSCD84 ③

nice mouse!
~~determine~~ which direction to go.



A*
↳ UCS → using a heuristic $h(n)$ to estimate nearn
to goal
Admissible heuristic
$h(n) \leq h^*(n)$ ⟶ don't over estimat
estimated cost of ^ actual cost

If $h(n) > h^*(n)$ miss opt sol'n.

$h\cancel{(n) \neq \phi}$, $h(n) = \phi$
↕
$h^*(n)$

Search BFS/DFS.
↳ opt sol'n (action taken)
· actions have a cost (different)
UCS ⟵ Dijkstra's Method ⟵ shortest path on grap

They missing ⇒ how close ~~to th~~ are we to goal

search first down promising paths.

→ Question: Less search means
· Less nodes expanded (time)
· cost of agent actions!

A* → like UCS expands cheapest available action first.

$g(n) = d(n) + h(n)$

$d(n)$: distance (cost) from start

$h(n)$: estimate of distance to goal.

n_1, n_2: $d(n_1) < d(n_2)$, $h(n_2) \ll h(n_1)$

$h(n)$ — heuristic

· admissible:

$h^*(n)$ ← actural cost of getting to Goal from n.

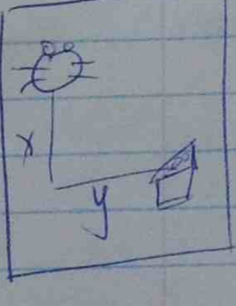
$h(n) \le h^*(n)$

if $h(n) > h^*(n)$ ← missing optimal solution.

· How to prove admissibility?

say $h(n)$ = your function of state

argue $h^*(n)$ (whatever it may be) can't be $< h(n)$



$h(n) = x(n) + y(n)$

$g(n) = \underline{d(n) + h(n)}$ add two parts.

$\max(h_1(n), h_2(n))$ ✓

h_1, h_2 admissible. lowest $h_1(n) = \phi$.

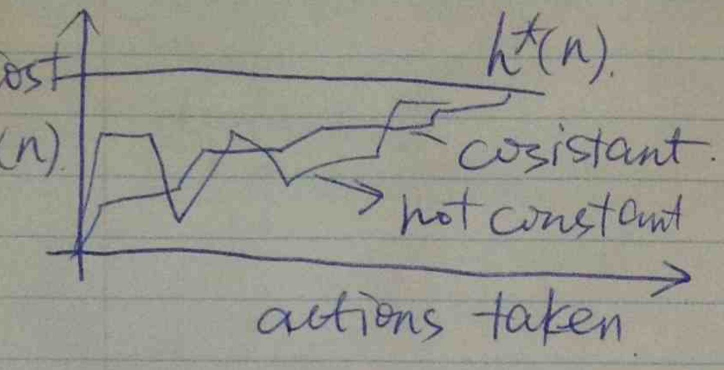
largest $h_2(n) = h^*(n)$

$h^*(n)$

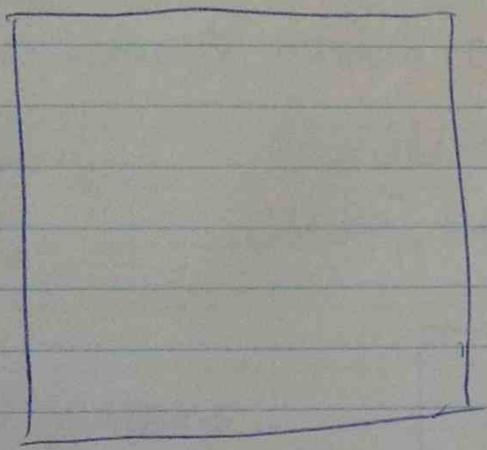
ϕ

CSCD84 (4)

Cosistent means $g(n)$ increases monotonically



Still admis.
not consistant



A^* no kitty

CSCD18 (5)

Constaint satisfaction

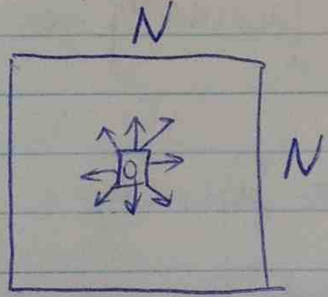
- subset of search problems
- Set of variables x_i – can take values from some domain D
- Set of constrants – single, pairs, sets

Ex:
Scheduling
variables: times for t_i course lectures to take place
constraints: which course can't be at same time.

Domain
9:00
10:00

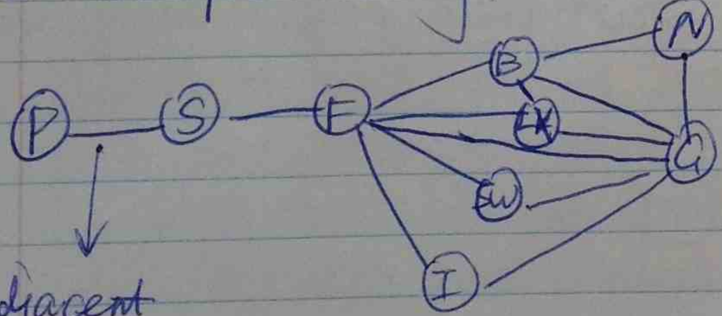
variables
CSCB07, B09.

Ex:
N-queens
N queens, can't attack each other



solve with search.

Ex. Graph coloring.



adjacent contries.
↑
constraints

Domain = {R, G, B, Y}

Types of CSP

a) Finite Domain — discrete.

n variables, if domain size d. $O(d^n)$

• Boolean SAT.

b) Infinite domains

c). Continuous vars. → Linear programming

Types of constraints

unary ⇒ Spain is Red

binary → $S \neq F$

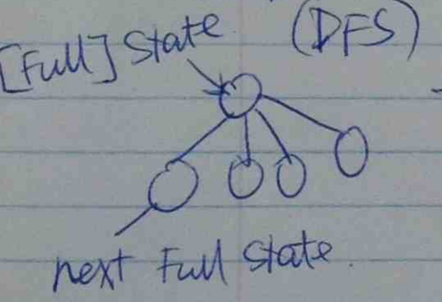
higher order → hard.

soft constraints → penalty for breaking

Solving CSPs

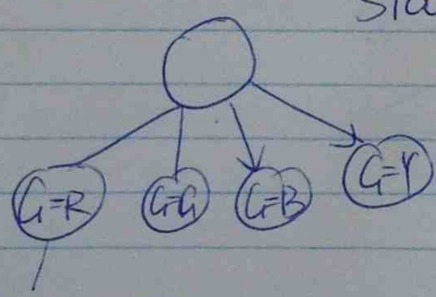
↳ Backtracking ~~track~~ Search.

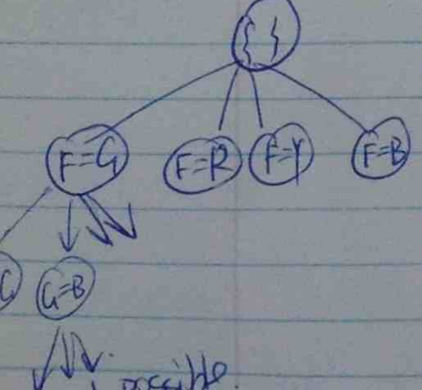
(DFS)



— Full State is a solution.

Start with empty assignment.





not possible.

then backtrack

to F.

Do we ever create entire search tree?

CSCD84 6

CSPs – Recursive Backtracking

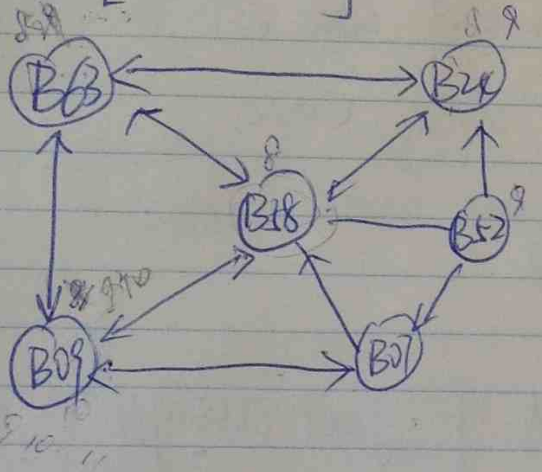
variables → X_i nodes

edges → constraints

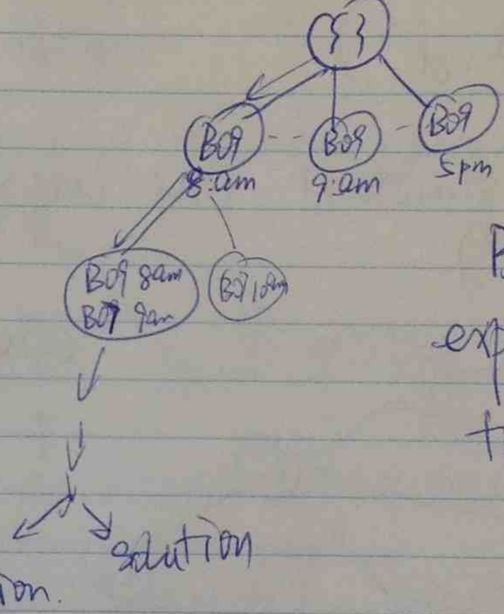
X_j

{ } ← tree
↳ contains some partial valid assignment of variables

values from a
→ [domain]



choose a variable



DFS

BFS is going to expand the whole tree.

Improving Backtracking Search

② choose order of variables carefully
. which variable has most constants
↳ places more constraints on remaining variables

① choose first variable with fewest possible values left.

③ Once you choose a variable → Choose the least constraining value first
max chance of <u>success</u>.

Arc Consistency

↳ Once you assign a value to one variable, go check that all remaining variables have at least one valid value left.

$O(d^n)$ Standard CSP.

n — # of vars

d — size of ~~vars~~ domain

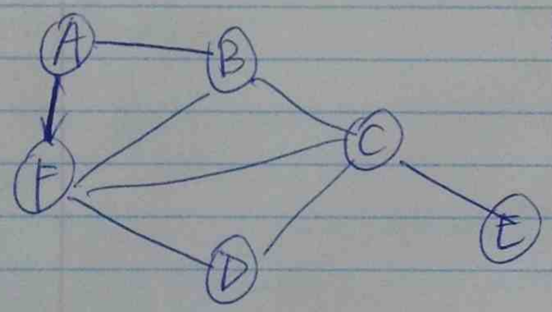
break problem into subproblems of size c.

$O(\frac{n}{c} \cdot d^c)$ $n=80$, $d=2$, $c=20$.

CSP - tree structure (no loops)

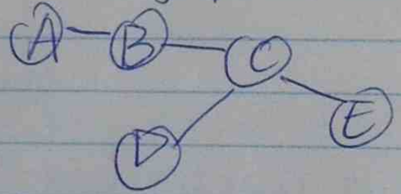
$O(nd^2)$

nearly T.S. CSP



what if just assign F a value?

F = RED. (graph coloring)



Tree

For each value of F, solve remaining T.S. CSP

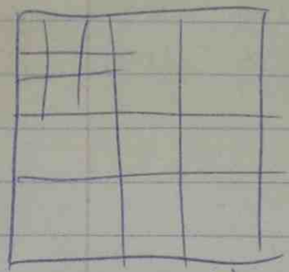
$O(nd^2) \times d$.

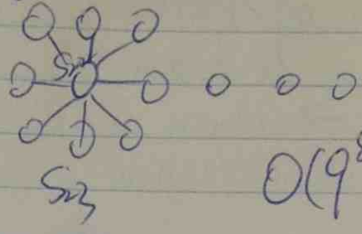
Iterative Solutions

* sub optimal — good enough.

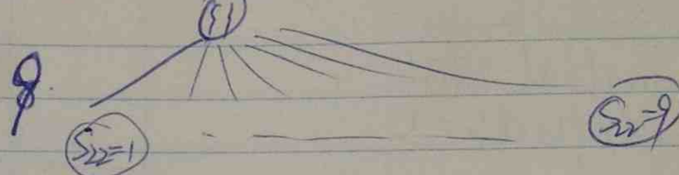
CSCD84 (7)

Variables: empty squares (81). domain 1-9.





$O(9^{81})$



backtracking search.

step (1) choose var $S(2,2)$

worst case branching factor anywhere is 9.

8 S_{32}

S_{99} (9) options.

Reasonable solution in a reasonable time

Iterative Method $\boxed{\text{Local Search}}$

- Always complete assignment (may not satisfy constraints) (Random Initial state)
- Operators — reassign values to variables

at each step

choose 1 variable randomly

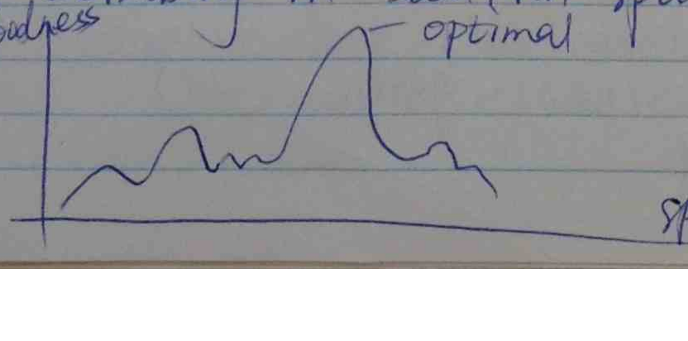
choose different value / randomly

check constraints

if new better than old : keep

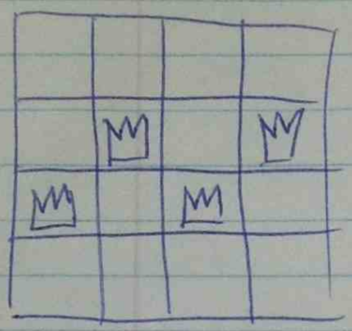
otherwise — move on

- Hill climbing in solution space



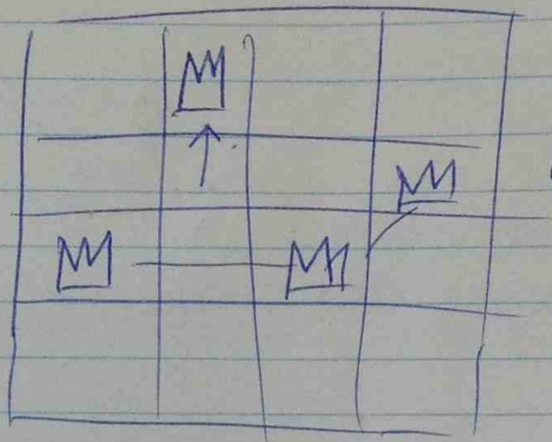
Local max are a problem

4 Queens



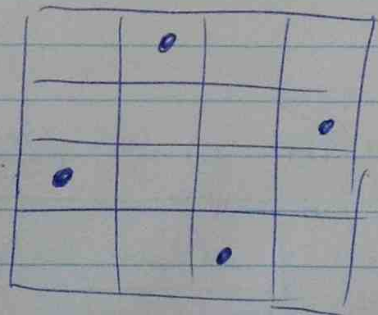
initial state
conflicts=5

operator
choose 1 Queen randomly.
→ then move it on the same column



conflicts=2.

solve n-queens for
n=10,000,000
in nearly constant time



→ Dealing with local max/min

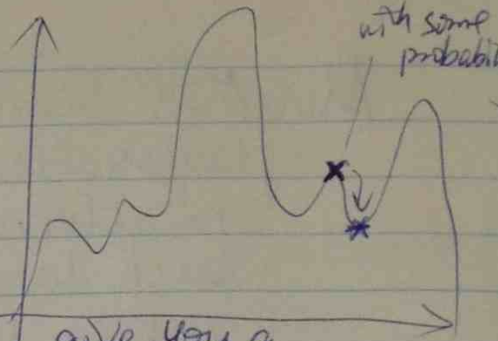
→ Random re-starts keep best sol'n
→ Simulated Annealing
(Deterministic) "

D.A Search: → Initially random state
→ start a Temperature = k
Loop
chang 1 variable at random
ΔE = goodness(old) − goodness(new)
if goodness(new) > goodness(old)
keep.

CSCD84 8

otherwise:

Accept with probability proportional $e^{-\Delta E/T}$



$\longleftarrow$ make $T = T * decay \rightarrow [0,1]$

↓ happen outside "if".

give you a change to a worst sol'n, to get a better one.

GAME PLAYING

↳ Adversarial 2 player game.

- pong
- chess
- checkers
- M vs C.

- Utility function
  - how do we represent game configurations?

X		O
X	O	
X		

- - -

player 1 +100

2 −100

game states

scoring game configurations.

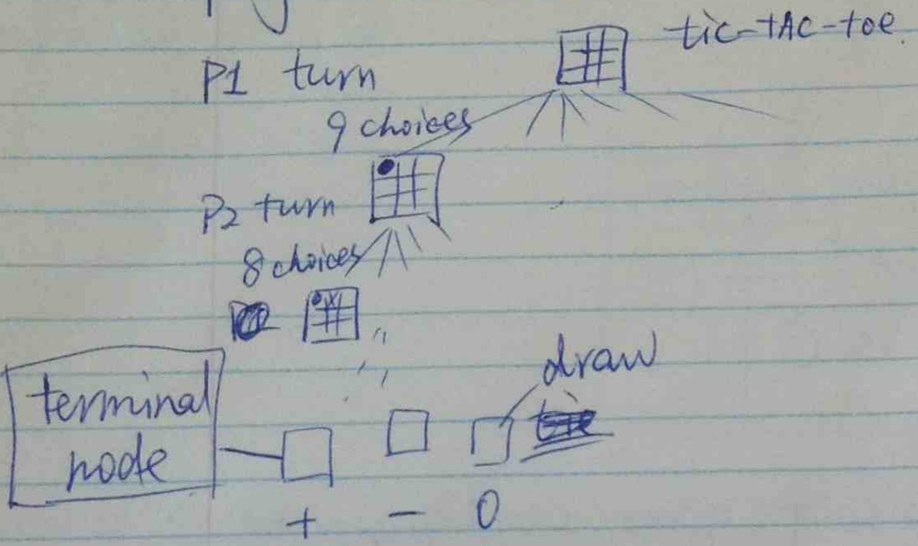
General Strategy

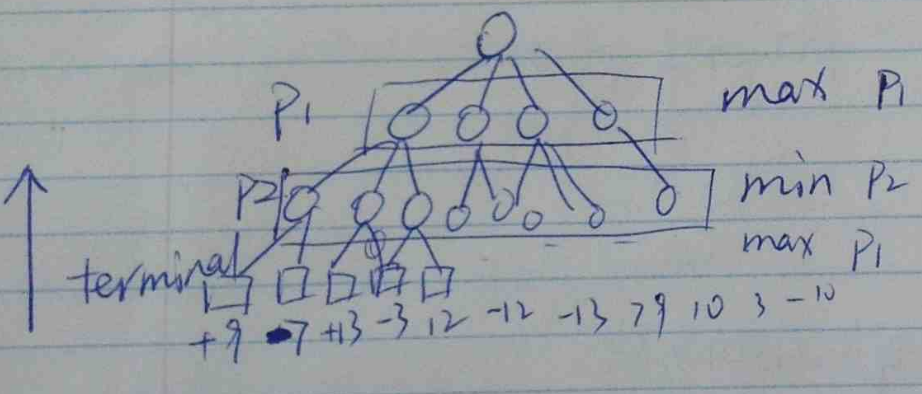
P1 — Choose sequence of moves that maximize util

P2 — Choose sequence of moves that minimize utilit

MiniMax Search

- players alternate
  - ↳ alternating min/max
- each level in search tree contains all possible moves for 1 player at 1 turn.
- players choose move with largest minimax value.



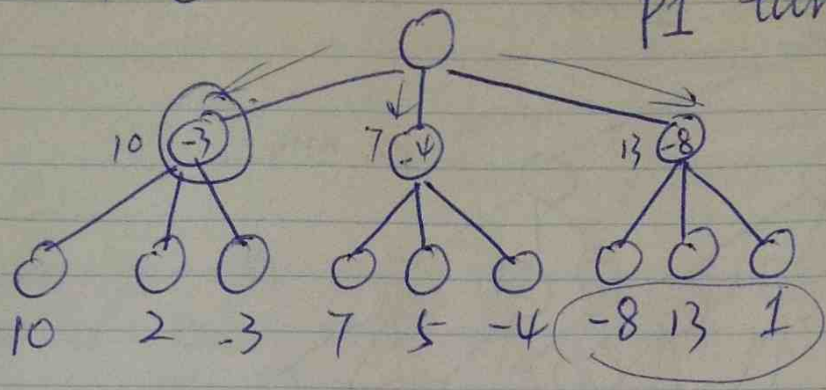


Minimax

- Create starting node
- During the game
  - Determine turn
    - max() min()
  - Search for successors to the current node up to a $\boxed{\text{pre-defined depth}}$
  - Evaluate each of the $\boxed{\text{leaf}}$ nodes
  - Utilities propagate up the tree alternating min/max

CSCD84 (9)

P1 turn. max.



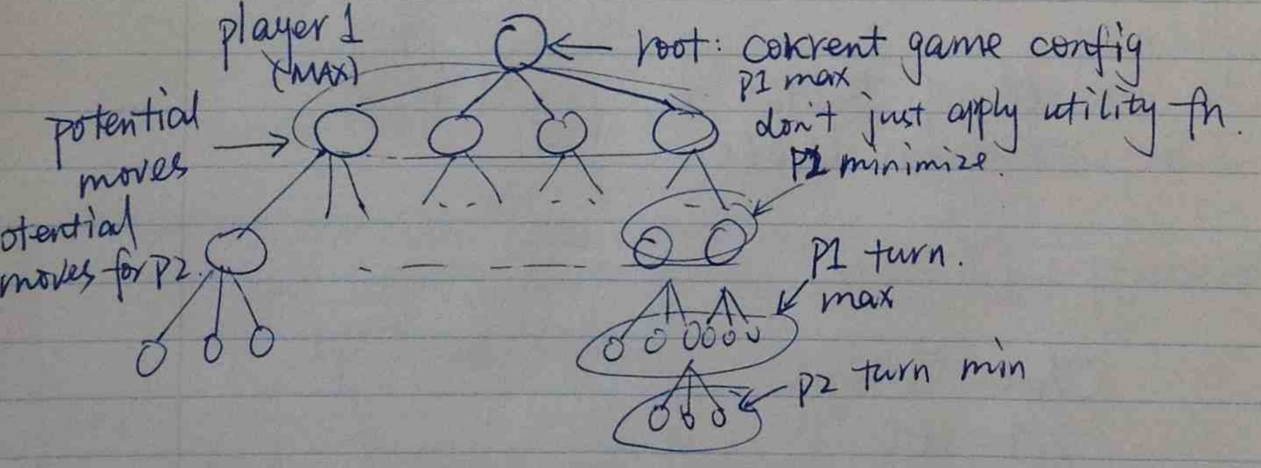
then choose

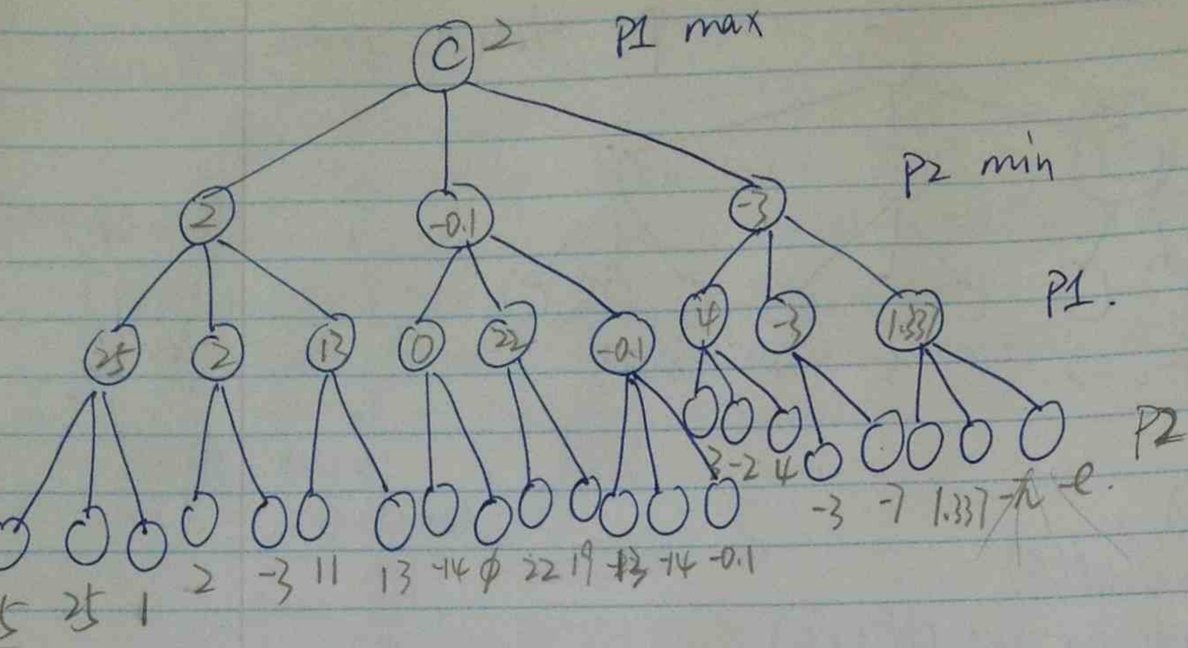
if 8 13 1

A3

```
minimax(s)
  if terminal(s)
      return utility(s)
  for i in successors(s)
      v[i] = minimax(i)

  if s.type = max
      return max(v[i])
  else
      return min(v[i])
```





Search depth makes a difference

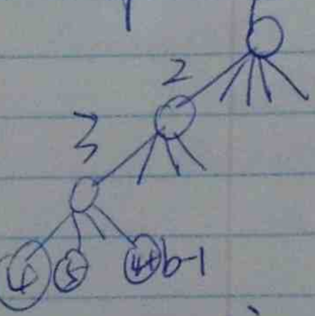
\# of turns

1-ply * # of players

Properties of Minimax

- Time complexity
  Tree with branching factor b, and I'm looking m levels $O(b^m)$

- Space Complexity $O(mb)$



Chess $b \approx 35$
$m = 100$
35^{100}

novice player = 4 ply
master → 8 ply
Grandmaster → 12 ply

CSCD84 10

Utility function: → return goodness of configuration
(+) favors player 1
(−) favors player 2.

In practice, the only "certain" utility is at end-nodes
partial games → tricky (dont know) ~~how it~~

think of typical approach.
— Select a number of features

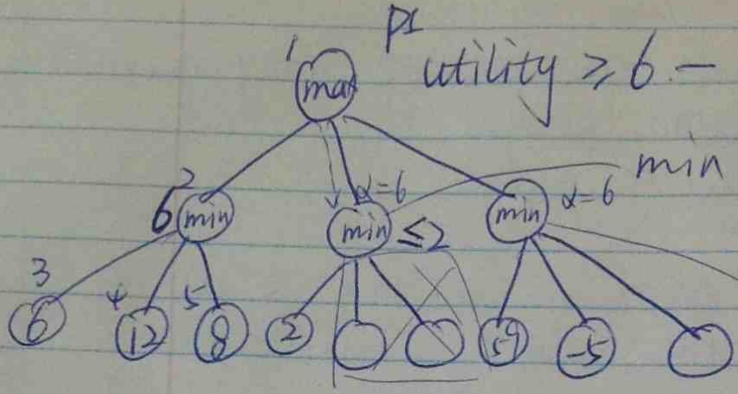
— Linear combination

$$U = \sum_{i=1}^{N-F} w_i f_i$$

$$\begin{aligned} U = \; & 200(K-K') \cancel{+9(Q-Q')} \\ & +9(Q-Q') \\ & +5(R-R') \\ & +3(B-B') \\ & +3(Kn-Kn') \\ & +1(P-P') \\ & + \text{other stuff.} \end{aligned}$$

Reducing Search Tree complexity
- Prune (cut) parts of the tree that cannot be part of the optimal sequence of moves.

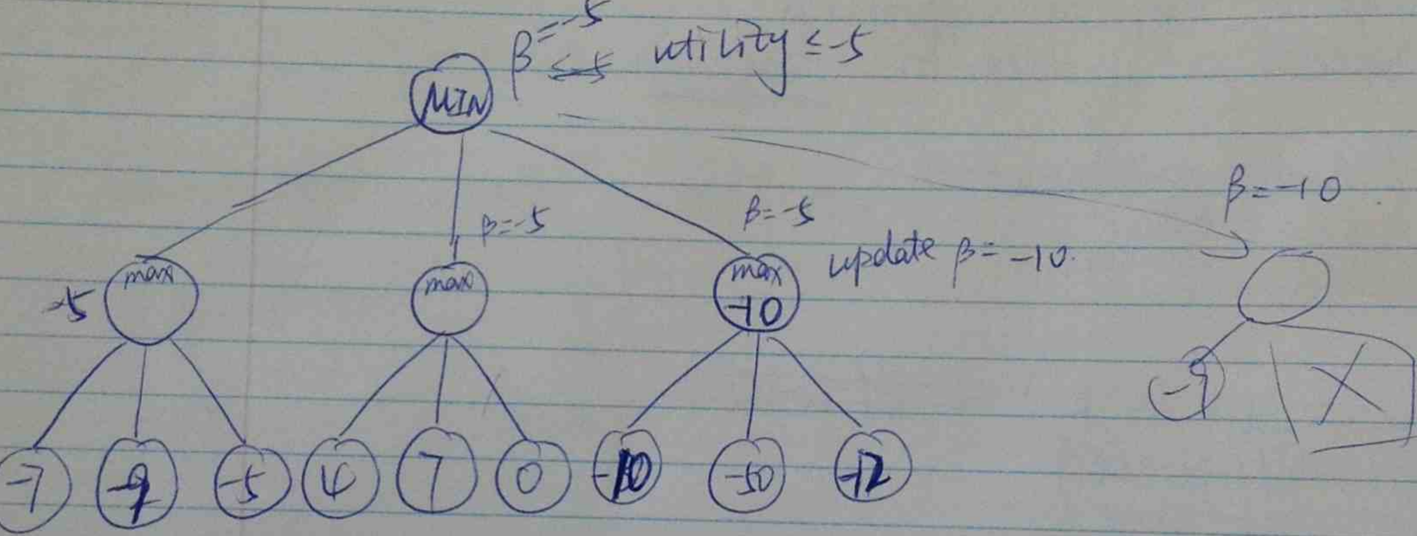
α-β pruning — Branch & Bound.



utility ≥ 6 — alpha — set at MAX nodes

min node prunes as soon as a value is found that is less than α.

utility ≤ -5.



— chop branching factor.

$b \rightarrow \approx \sqrt{b}$.

Search twice as deep.

CSCD84 11

Maximizing Expected Utility = rational agent

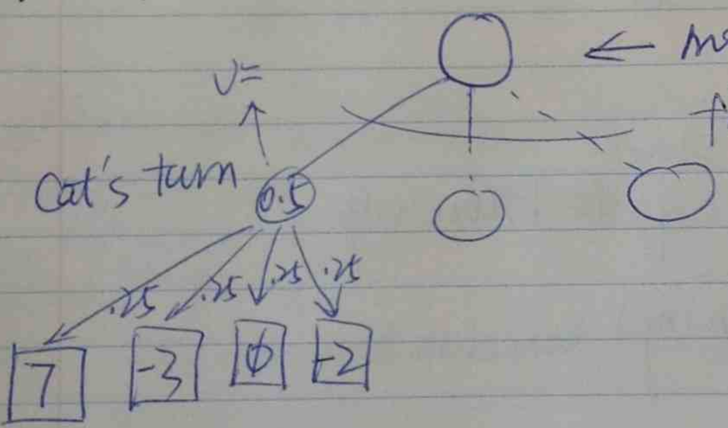
MiniMax.

- Players are rational / play optimally.
  - Modify MiniMax, instead of [utility] → Expected utility.

for games with chance

Actions map to multiple results. ←
- → randomness (dice, card shuffling)
- → insufficient evidence
- → unmodeled variables
- → noise

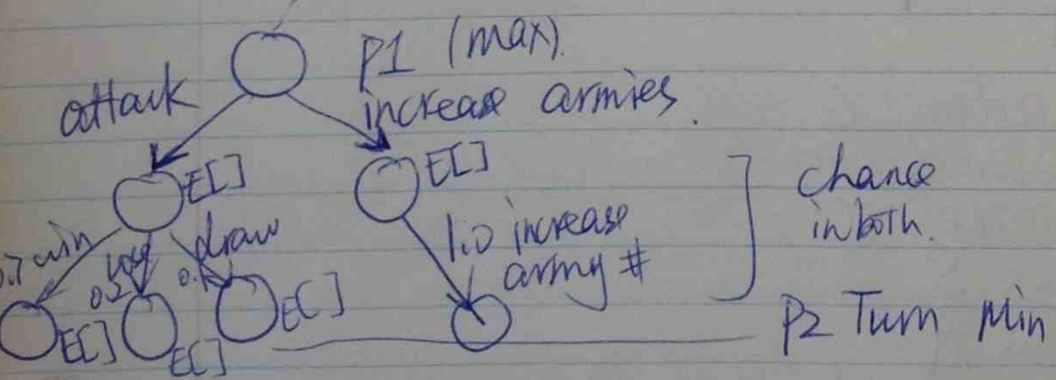
Example — mouse vs random cat.



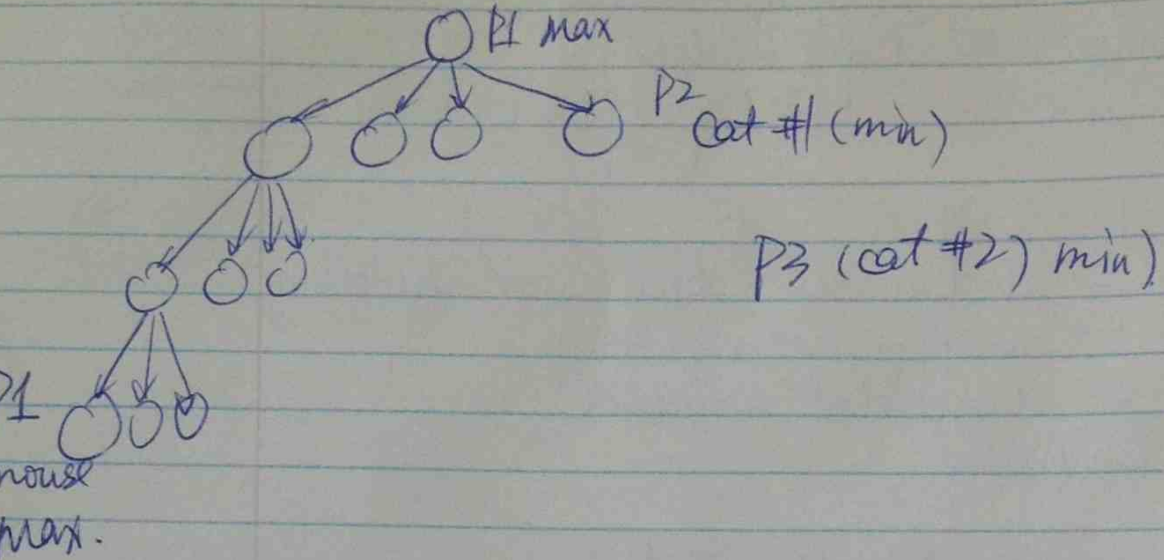
Expectimax.

$U = 0.25(7-3+0-2) = 0.5$

Risk → ExpectiMiniMax.



1 mouse 2 cats



Non zero-sum games — scrabble

Agents know lots
↳ from the start
↳ Possibly missing / incomplete knowledge
→ Learn from experience
(Reinforcement Learning)

Idea:
→ keep track of statistics of the utility (reward) of ~~kein~~ performing certain actions in certain states

- Agent receives input i
  (state S)
- Agent carries out action a.
- Agent receives reward (+/−)
  ↳ new state S′

Model — set of states S
set of actions A
reinforcement signal (reward)

CSCD84 12

Task for agent?

↳ Come up with a policy π mapping states to actions, s.t. rewards are maximized over time

bad: Environment changing stats not apply. Encode to apply stats.

Transition $(s, a) \rightarrow s'$ – (chance is involved)

Transition function $T(s, a, s') = p$ (probability)

Stats: on states ✓

on action ✓

on transitions → assume given.

on rewards ✓

Maximize Expected rewards.

$$E\left(\sum_{t=\phi}^{\infty} \gamma^t r_t\right)$$

γ → discount factor $(\phi, 1)$

(I don't know whether can survive that long. $\gamma^t \downarrow$ when $t \uparrow$)

r_t → Reward obtained at time t.