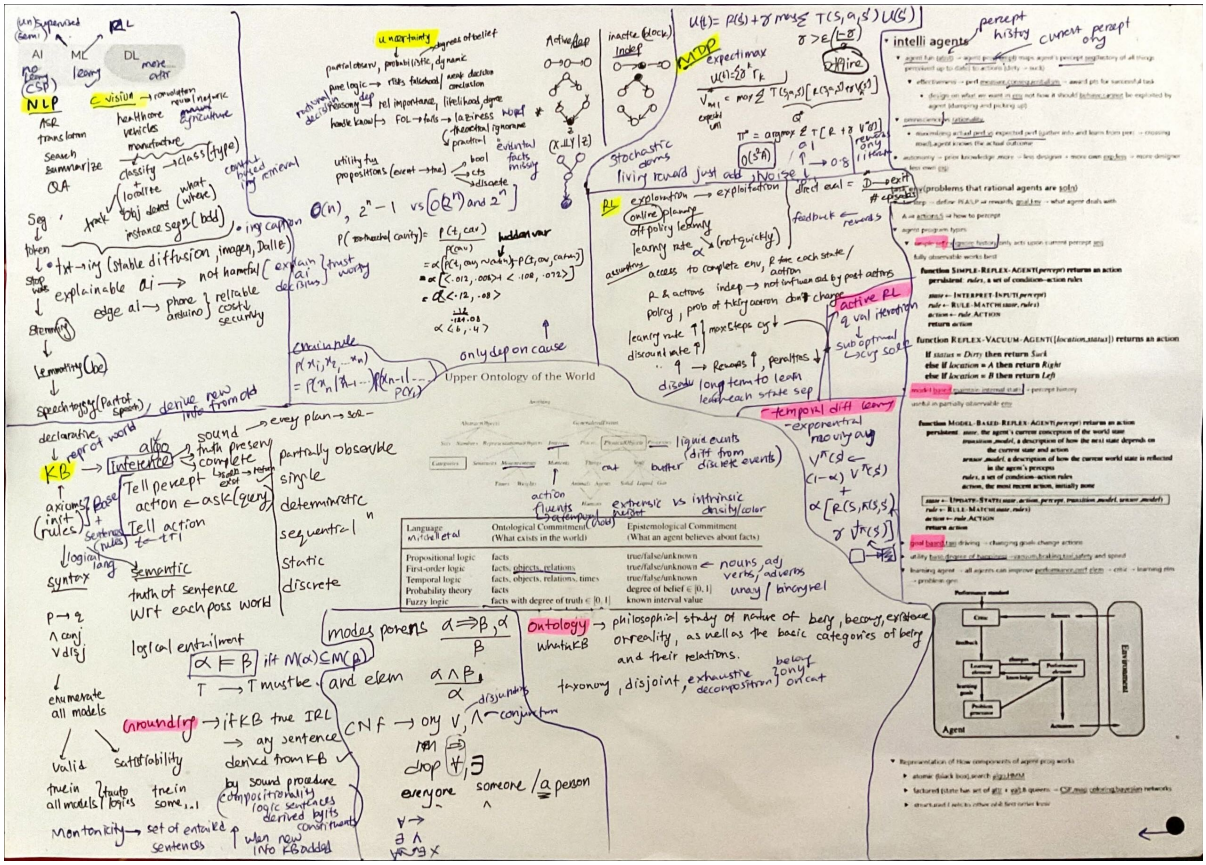
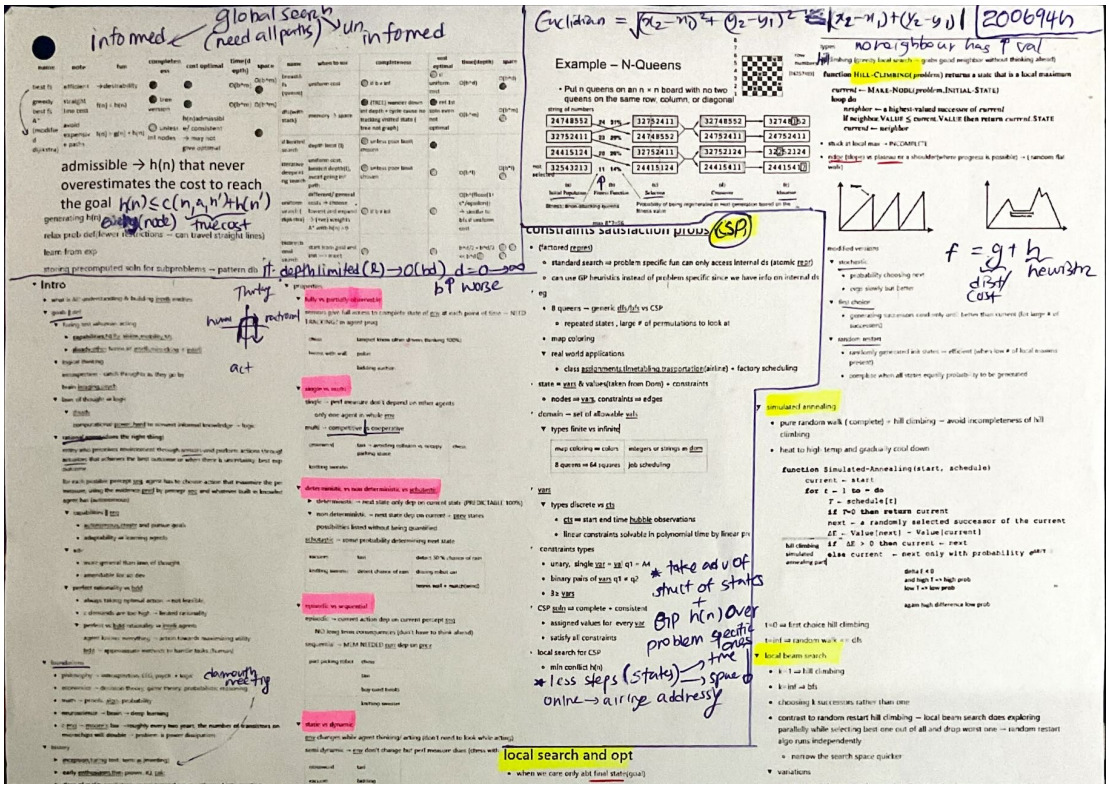
Ai ss

Sunday, 26 February 2023 11:38





- expert [illegible], lawyer
- probabilistic reasoning
  - [illegible] → based on math models/theory and real data → speech, chr. handwritten recognition
  - bayesian networks → rigorous, efficient formalism for repr uncertain knowledge; practical algo for probab reasoning
- big data
- deep learning
- state of art → recent stage of dev
  - publication → conference → industry
  - robots → sophia, boston dynamics
  - alexa, siri → speech recognition
  - google translate → chr. rec., machine translation
  - autonomous vehicles → many constraints planning and scheduling, tesla
  - expert systems: specialized knowledge operates in well defined domains only } limitations
  - privacy vs cyber attacks

chess

buy own books

- discrete vs cts

finite number of percepts and actions that can be performed within a number of DISTINCT states + cts time problem

chess — taxi

shopping

bidding

agent=archi + program

percepts → actions

Planning

appl: mobile robots

sw test case: gen

transportation agents

current state → ? → goals

seq of actions schema

action name

Actions (Fly(p, from, to))

must be T: Precond: At(p, from) ∧ plane(p) ∧ airport(from) ∧ Airport(to)

effect: ¬At(p, from) ∧ At(p, to)

→ ADL (expressive)

every literal not mentioned remains unchanged

↳ frame prob: explicitly specify which cond not chged by actions [what is left unchanged when action is performed]

fluents → dep on time (condition change)

classical planning → changes only when agents act

PDDL

dom def

assumptions:

- first order
- ground (no vars)
- fun free; (+)ve literals, ∧
- ground → no complex
- closed world assump → not in KB → false
- unique name

planning Algo

state space search

→ Partial order planning (POP)

progression:

- fwd
- bwd (regression) ← goal → consistent (dom ind)
- dom specific
- dom indep → count diff facts
- heuristic → approach soln, not guaranteed an optimal/perfect, but sufficient for immediate goal, quick/cheap soln

dom not well understood

(est) (cost)

branching (subgoal) lifting strips

- cannot avoid irrelevant actions
  - good h(n)
- b9 (blocks world)

quick vs cheap

STRIPS Language	ADL Language
Only positive literals in states: $Poor \wedge Unknown$	Positive and negative literals in states: $\neg Rich \wedge \neg Famous$
Closed World Assumption: Unmentioned literals are false.	Open World Assumption: Unmentioned literals are unknown.
Effect $P \wedge \neg Q$ means add P and delete Q.	Effect $P \wedge \neg Q$ means add P and $\neg Q$ and delete $\neg P$ and Q.
Only ground literals in goals: $Rich \wedge Famous$	Quantified variables in goals: $\exists x\, At(P_1, x) \wedge At(P_2, x)$ is the goal of having P_1 and P_2 in the same place.
Goals are conjunctions: $Rich \wedge Famous$	Goals allow conjunction and disjunction: $\neg Poor \wedge (Famous \vee Smart)$
Effects are conjunctions.	Conditional effects allowed: when P: E means E is an effect only if P is satisfied.
No support for equality.	Equality predicate $(x = y)$ is built in.
No support for types.	Variables can have types, as in $(p : Plane)$.

- don't keep track of path (if we reach the goal the path is trivial)
  - 8 queens, travelling salesman prob
  - crop planning, hill climbing
- adv
  - space efficient & soln in a big infinite space
    - reasonable solns in a large (inf) state problem where other a
  - suitable for optimization prob → opt a objective fun (== h(n))
  - begin with complete config (8 queens, IC design)

searching → prob solving agents

plans ahead, consider seq of actions to find goal state(soln = need a path)

atomic repr (black box)

- problem solving process
  1) form goal → limit objectives
  2) form problem → def → states(nodes) + actions(edges)
  3) search through until soln →
  4) exe one at a time
- search prob components / terminology
  - state space (set of all possible states in env) → GRAPH, init vs goal (1 or inf)
  - actions → actions(katunayaka) = (to negombo, to colombo)
  - transition model → desc what each act do RESULT(kat , to neg) = neg
  - action cost fun
    - action_cost(katunayaka , to negombo, negombo) = 8 km or 20 min
    - action_cost(state, action, result) = cost
- path(seq actions to soln) → soln → optimal soln(lowest cost) > assume positive
  - how to know what to expand in frontier search tree from interior nodes →
    - redundant paths (find lowest cost one)
    - repeated states (meeting top nodes)
- measuring prob solving performance (evaluation criteria)
  - completeness → guarantee to report if found soln else report no soln found
  - cost optimal → lowest cost soln
  - b,d → m (everything is max count)
    - space
    - time

- stochastic beam search → probability choosing k successes → to avoid concentrating on a local area

Beam search → limit frontier; incomplete / suboptimal but eff with reasonably good soln

- genetic algo

variation of stochastic beam search, N=1 ⇒ random walk

- successor(child) generated combining two parents
- init population(k states) → fitness/eval fun → selection → mutation → crossover
  - init pop representation
    - individual → string, each elem called gene(usually bin)
    - selecting strat
      - random
      - probability dep on fitness fun
      - done with replacement to make sure fit individuals reproduce mult times
    - reproduction
      - random pairing
      - define random crossover pts
      - each gene → altered by mutation
- adv(+ ve points) vs disadv
  - no convincing evidence that GA is better than local search ⇒ useful on some/small set of problems

random(crossover) ⇒ soln that local search couldn't find	tunable parameters huge
connected to human evolution	replicate part one prob to another

Genetic Algorithms

```
function GA (pop, fitness-fn)
Repeat
   new-pop = []
   for i from 1 to size(pop):
      x = rand-sel(pop,fitness-fn)
      y = rand-sel(pop,fitness-fn)
      child = reproduce(x,y)
      if (small rand prob): child ← mutate(child)
      add child to new-pop
   pop = new-pop
Until an indiv is fit enough, or out of time
Return best indiv in pop, according to fitness-fn
```

```
function reproduce(x,y)
   n = len(x)
   c = random num from 1 to n
   return:
append(substr(x,1,c),substr(y,c+1,n))
```

Local Search for CSPs

```
function MIN-CONFLICTS(csp, max_steps) return solution or failure
  inputs: csp (a constraint satisfaction problem),
      max_steps (the number of steps allowed before giving up)
  current ← an initial complete assignment for csp
  for i = 1 to max_steps do
      if current is a solution for csp
      then return current
      var ← a randomly chosen, conflicted variable from VARIABLES[csp]
      value ← the value v for var that minimize CONFLICTS(var,v,current,csp)
      set var = value in current
  return failure
```

take min conflict