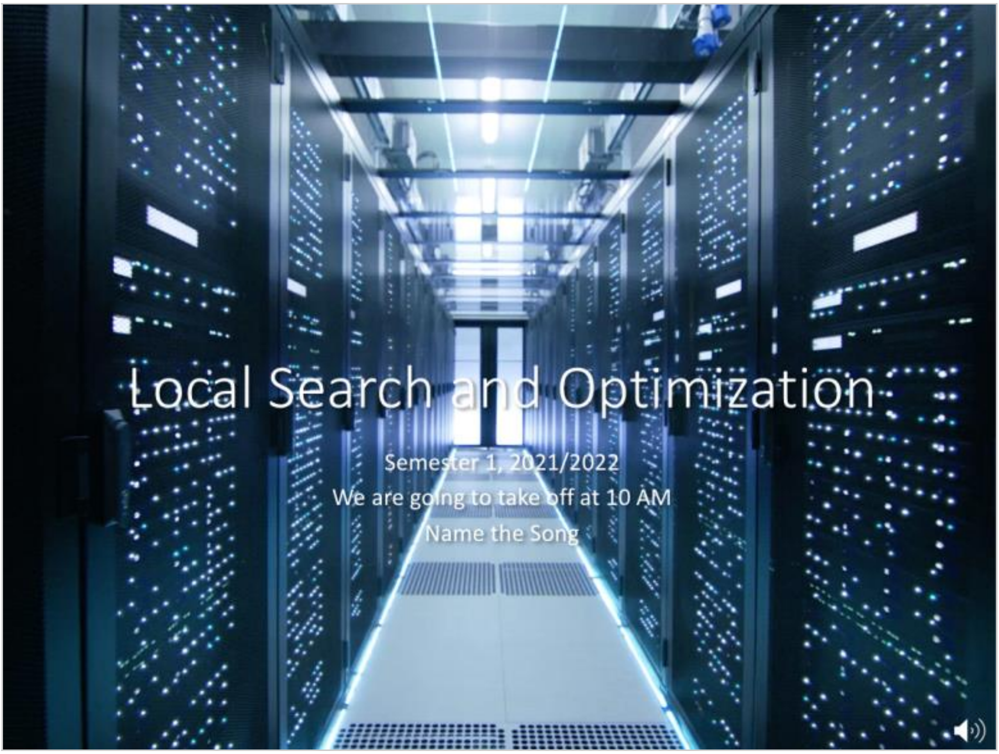
Local Search

Sunday, 5 September 2021 2:03 PM



Logistics

- Lecture Notes: cs3243-notes.github.io
- Grading: **I DO NOT GRADE ON CURVE**
  - Cut-offs are based on difficulty levels. You are only competing with yourself.
  - My ideal scenario: Everyone gets A
  - Realistic: Only A and B barring exceptions
- (Relaxed) Late Policy:
  - Tutorial Assignments: One day late allowed with 30% penalty
  - Projects: 10% penalty per day of delay upto 3 days.
  - No exceptions: We are on very tight schedule
- We make minor changes in lecture notes based on questions or if we notice typos; but no conceptual change.
- Projects: You are free to implement data structures or code optimizations.

2

The Story So Far

- Agents seek to find the minimum cost path to the goal.
  - Uninformed Search: without usage of any information about the cost to reach goal state
  - Informed Search: have “credible” (admissible/consistent) information about the cost to reach goal state

Today

- Problems where path to the goal is irrelevant

  Agent is interested in reaching the goal state.

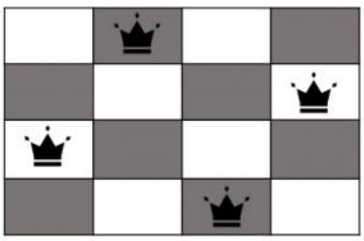
  (Secret: “Sometimes Ends justify the means”)

- You will learn learning one of the most powerful (and amazingly simple) algorithmic technique we know in Computer Science.

3

The N-Queens Problem

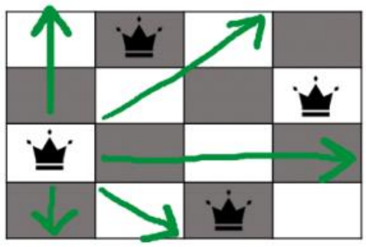
Place N chess queens on NxN chessboard such that no two queens threaten each other.



N = 4

The N-Queens Problem

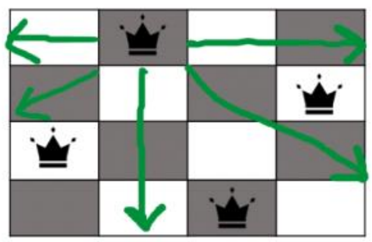
Place N chess queens on NxN chessboard such that no two queens threaten each other.



N = 4

The N-Queens Problem

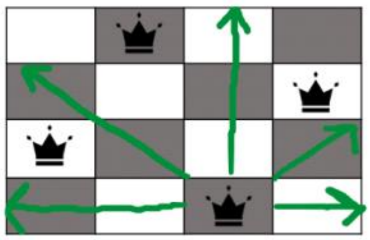
Place N chess queens on NxN chessboard such that no two queens threaten each other.



N = 4

The N-Queens Problem

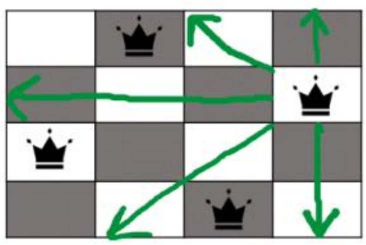
Place N chess queens on NxN chessboard such that no two queens threaten each other.



N = 4

The N-Queens Problem

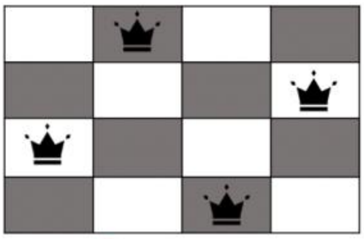
Place N chess queens on NxN chessboard such that no two queens threaten each other.



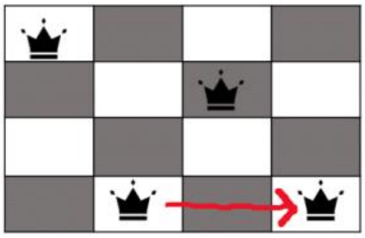
N = 4

The N-Queens Problem

Place N chess queens on NxN chessboard such that no two queens threaten each other.



N = 4

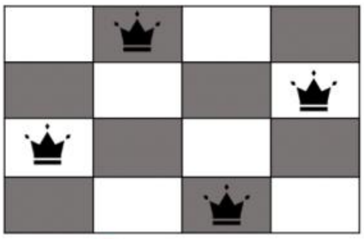


N = 4

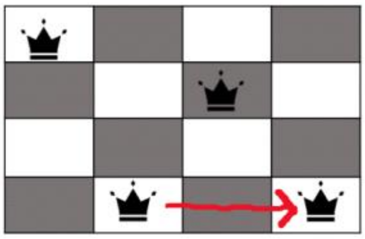
The N-Queens Problem

Place N chess queens on NxN chessboard such that no two queens threaten each other.

☆ Easy to check but not to find



N = 4



N = 4

Module Scheduling

1. Two modules can not be scheduled in the same room at the same time.
2. The room where the module is scheduled should be large enough to handle projected registration.
3. Certain modules should be scheduled in evenings as much as possible.

……

……

…..

All we care about: A schedule that meets all the requirements

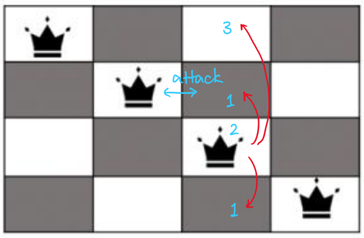
Finding a Solution

- Every column must have exactly one queen, hence we place one queen in every column and only move queens along columns.

Finding a Solution

- Every column must have exactly one queen, hence we place one queen in every column and only move queens along columns.
- Start from a random position
- Move to a better position

→ # of queens that attack each other



Initial

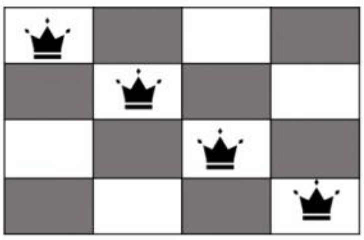
Now:
$_4C_2 = 6$ pairs (3 if looking only at column 3)

Move queen in column 3 up by 1 space:

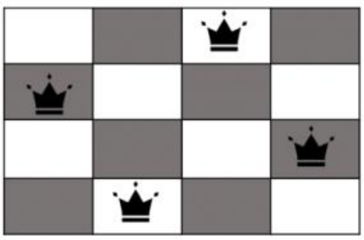
$_3C_2 + 1 = 4$
(1 looking only at queen 3)

Finding a Solution

- Every column must have exactly one queen, hence we place one queen in every column and only move queens along columns.
- Start from a random position
- Move to a better position



Initial



Solution

Abstracting the Problem

- S : Set of board states.

move anywhere within the same column

- N(s) : Neighbors of state s in S
- Val(s) : Values of a state s in S. → # of queens that attacked each other

We want to Val(s) to reflect "quality" of the state, in a sense of how close it is to the goal state.

Abstracting the Problem

- S : Set of board states.
- N(s) : Neighbors of state s in S
- Val(s) : Values of a state s in S.

We want to Val(s) to reflect "quality" of the state, in a sense of how close it is to the goal state.

- Val(s) = 0, if s is the goal state.
- Val(s) = # of pairs of queens that attack each other

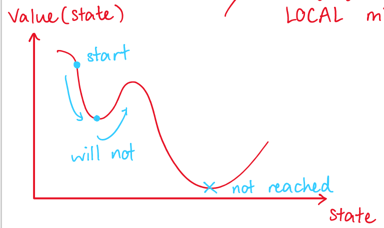
Hill Climbing

Algorithm 1 HillClimbStep(s)

$minVal \leftarrow \text{val}(s)$
$minState \leftarrow \{s\}$
for each u in $N(s)$ **do**
  if $\text{val}(u) < minVal$ **then**
    $minVal = \text{val}(u)$
    $minState = u$
return $minState$

☆ keep moving to a state where Val(next) < Val(current)

☆ only finds LOCAL minimum

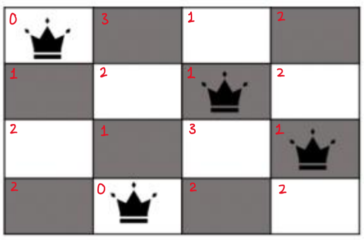


17

Limitations of Hill Climbing

- Hill climbing only allows moves to better positions
  - ↳ simple but flawed
  - ↳ INCOMPLETE

Initial



val = 1

Value = 1

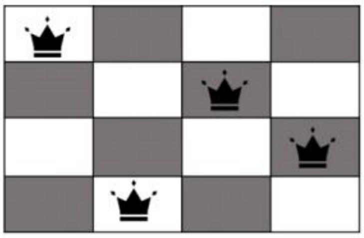
☆ No better moves

Allow Some Mistakes

- Hill climbing only allows moves to better positions
- What if we allow "mistakes" i.e. moves to worse positions?

Value = 1

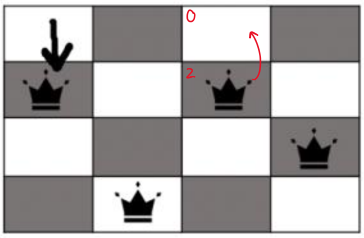
Initial



val = 1

Value = 2

Move 1

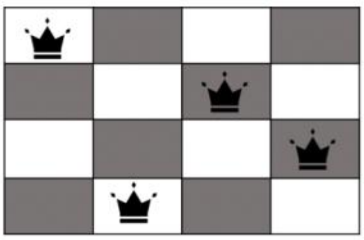


val = 2

Allow Some Mistakes

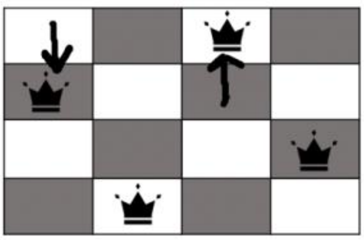
- Hill climbing only allows moves to better positions
- What if we allow "mistakes" i.e. moves to worse positions?
  ↳ ☆ mistake amongst neighbours, not any random state
    ↳ no. of possible states are very large

Initial

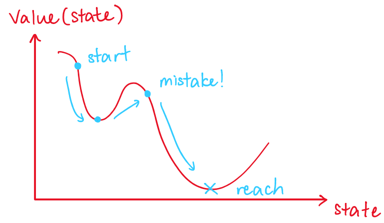


val = 1

Move 2



val = 0



Simulated Annealing

We want an algorithm that allows mistakes; allowing a mistake is to allow a move to a neighbor that has value higher value than the current state.

Story Time!

21

Simulated Annealing

- Condensed matter physics
  - Study of materials at low temperatures.
  - Spin Glass models
    - Atoms : have spin ± 1
    - μ_i : spin of atom i
    - E(c): Energy of configuration c
      - E(c) = $e^{\frac{-\sum_{(i,j)} J\mu_i\mu_j}{k_B T}}$
      - k_B : *Boltzman constant*
      - T : temperature
    - Probability of going from state c_1 to c_2
      - $\Pr[c_1 \to c_2] \propto e^{\frac{E(c_1)-E(c_2)}{k_B T}}$ → $T \to \infty$, $e^{\frac{E(c_1)-E(c_2)}{k_B T}} \to e^0 = 1$

$\Pr(c_1 \to c_2) \approx \frac{1}{n}$

- How do we reach the lowest energy state?
  - Material scientists : have a "cooling schedule"
  - Cooling schedule : "first have the high temperature and then slowly decrease the temperature"

$T = 1$, probability depends on $E(c_1) \to E(c_2)$

☆ When $E(c_1) > E(c_2)$, probability ↑. } generally goes to a better state

When $E(c_1) < E(c_2)$, probability ↓ but never 0 } but small chance of making mistakes

☆ How to ensure reaching to lowest energy state?

22

Simulated Annealing

How do we implement this algorithmically?

Kirkpatrick, Gelatt and Veechi (1983)

Replace $E(c) \to val(c)$

SimulatedAnnealing(*initialState*)

$C \leftarrow initialState$
for $t = 0$ to ∞ **do** — what are the neighbours?
  $C' \leftarrow$ PickRandomNeighbour(C) ↳ valid actions
  $T \leftarrow$ Schedule(t)
  if $val(C') = 0$ **then**
    return C' ☆ lowest value = solution
  if $val(C') < val(C)$ **then**
    $C \leftarrow C'$
  else — mistake
    $C \leftarrow C'$ with Probability $\propto \exp\left\{-\frac{val(C')-val(C)}{k_B T}\right\}$

can change

Annealing

9: else — mistake

10: $C \leftarrow C'$ with Probability $\propto \exp\left\{-\frac{val(C')-val(C)}{K_B T}\right\}$

can change according to the algo

- We terminate when $val(c) = 0$, but for some cases, the $val(goal)$ may not be defined.
  - Optimization problems seek to minimize $val(c)$
  - Schedule can be modified according to application

23

Lessons for Life

- Taking small steps works most of the time.
- Go in the direction where it gets better
- But beware of local minima
  - You gotta take some risky decisions

24