CS 486/686
Local Search

Yuntian Deng

Lecture 5

RN 4.1 · PM 4.7–4.8

Reminders

DUE TONIGHT **Tue May 26, 11:59 PM**

Chat 1 (Intro & uninformed search) on Chrysalis.

DUE THU **Thu May 28, 11:59 PM**

Chat 2 (Heuristic search & CSPs) — extended from Tue May 26.

Learning goals

- Describe the advantages of local search
- Formulate a real-world problem as local search
- Verify whether a state is a local / global optimum
- Describe strategies for escaping local optima
- Trace greedy descent, random restarts, simulated annealing, genetic algorithms
- Compare and contrast local-search algorithms

Why local search?

Classical search algorithms…

- … explore the space **systematically**. *What if the space is too large, or infinite?*
- … remember a **path** from the initial state. *What if we only care about the final state (e.g. CSP solution)?*

Solution: local search.

Local search: what it gives up

- No systematic exploration of the space
- No path back to the start
- No proof when no solution exists

Local search: what it buys you

- Fast, good-enough states on average
- Very small memory footprint
- Handles pure optimization (no goal test needed)

What is local search?

Start with a **complete** assignment to all variables. Take iterative steps to improve it.

State

A complete assignment to all variables.

Neighbour relation

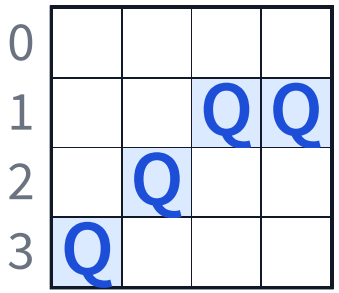
Which states can we step to next?

Cost function

How good is each state? Lower is better.

4-queens as local search

- **State:** one queen per column, any row — e.g. `(3, 2, 1, 1)`.
- **Initial:** random state.
- **Goal:** no attacking pair.
- **Neighbours:** *(A)* move one queen in its column; *(B)* swap two queens' rows.
- **Cost:** number of attacking pairs.



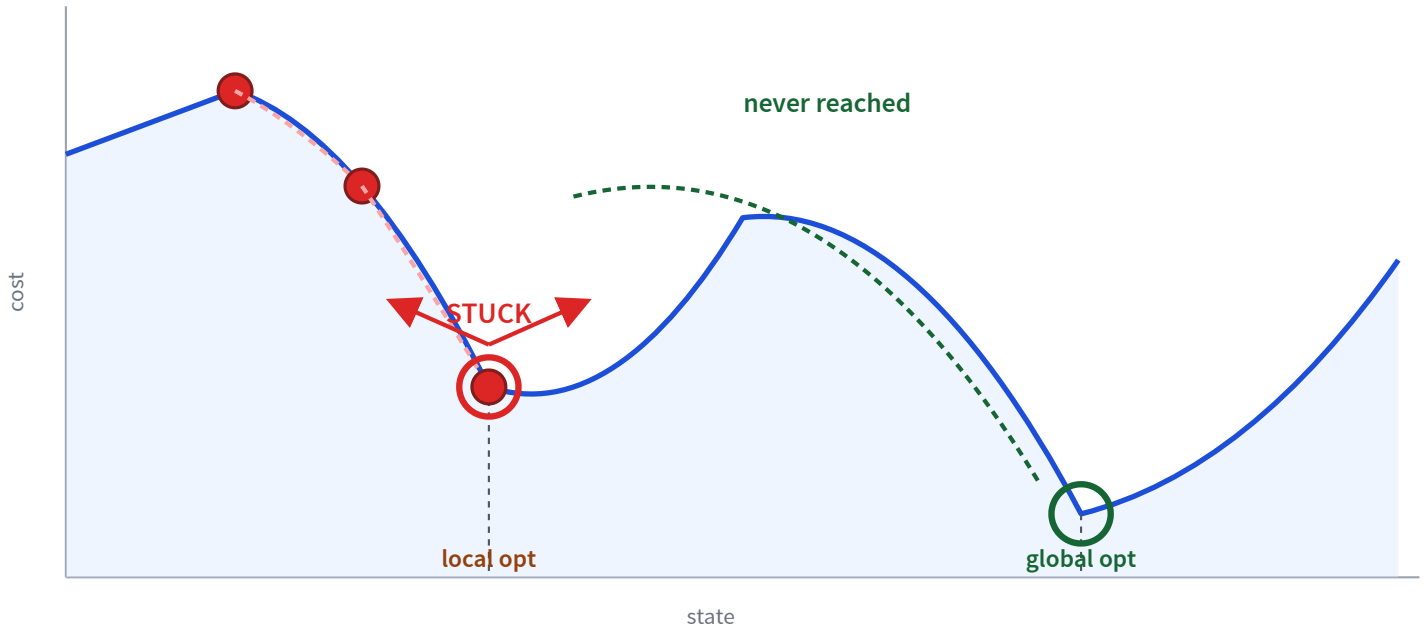
`(3, 2, 1, 1)`, cost = 4.

Greedy descent

a.k.a. hill climbing

- **Start** from a random state.
- **Move** to the neighbour with the lowest cost — *if it improves on the current state*.
- **Stop** when no neighbour is better.

Greedy descent on a landscape



Greedy descent: in a sentence

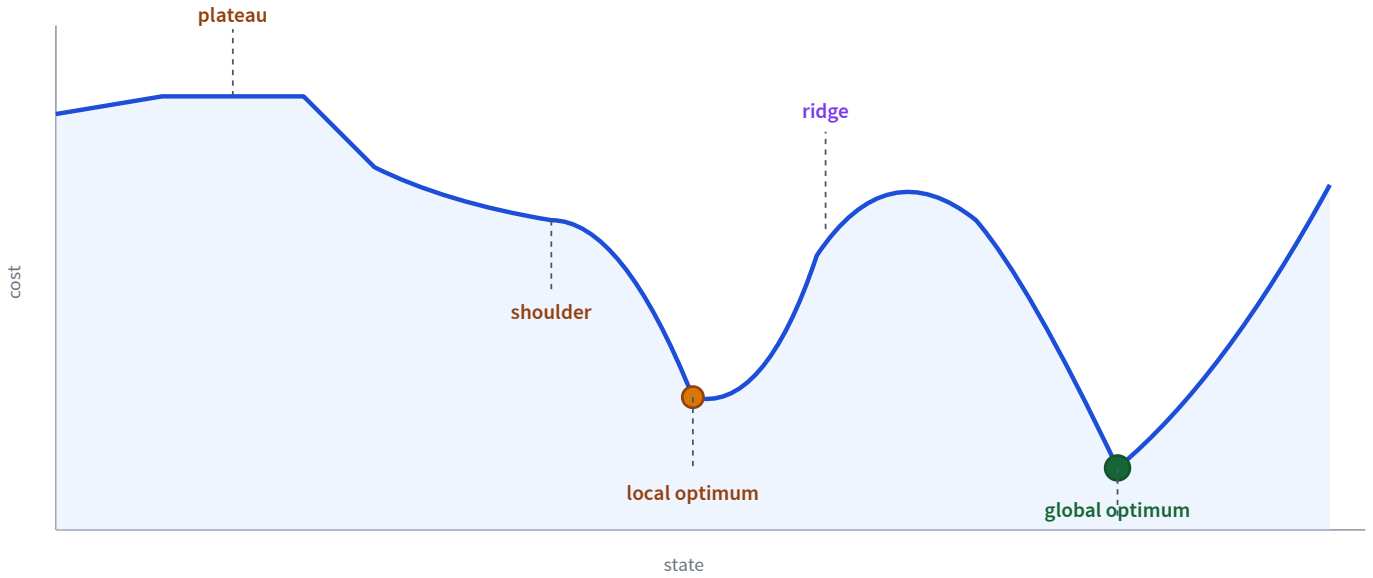
> *Descend into a canyon in a thick fog.*

- **Descend:** always move to the best neighbour.
- **Thick fog:** only see immediate neighbours.

Fast in practice — but only finds **local** optima.

State-space landscape

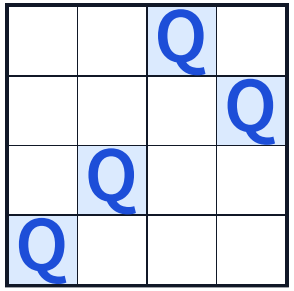
Local optimum: no neighbour has strictly lower cost. **Global optimum**: lowest cost overall.



A flat region with no slope is a *plateau*; a flat region at the top of a slope is a *shoulder*.

Q: local or global optimum?

Q. Consider neighbour relation **B**: swap the row positions of two queens. Is this state a local / global optimum?



State $(x_0, x_1, x_2, x_3) = (3, 2, 0, 1)$. Cost = 2.

A. Local optimum and global optimum.

B. Local optimum, but not global.

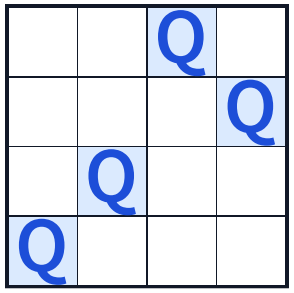
C. Not a local optimum, but it is the global optimum.

D. Neither a local nor a global optimum.

D — neither. Swap x_0 and x_2: new state $(0, 2, 3, 1)$ has cost 1 (better). So not a local

Q: local or global optimum?

Q. Same state, but now neighbour relation **A**: move *one* queen to another row in the same column.



State $(3, 2, 0, 1)$. Cost = 2.

A. Local optimum and global optimum.

B. Local optimum, but not global.

C. Not a local optimum, but it is the global optimum.

D. Neither a local nor a global optimum.

B — local optimum, but not global. Every single-queen move keeps or worsens the cost (try

Escaping flat local optima

Sideway moves

Accept neighbours of *equal* cost. Cap consecutive sideways to avoid loops.

Tabu list

Forbid recently-visited states — short-term memory.

Sideway moves help — a lot

8-queens: ≈ 17 million states.

Algorithm	% solved	Steps to success	Steps to failure
Greedy descent	14%	3–4	3–4
Greedy + ≤ 100 sideway moves	94%	21	64

A tiny tolerance for flat moves turns 14% into 94%.

Choosing the neighbour relation

Aim for a **small incremental change** to the variable assignment.

Bigger neighbourhoods

- Compare more nodes per step
- More likely to find the best step
- Each step takes more time

Smaller neighbourhoods

- Compare fewer nodes per step
- May miss the best step
- Each step is fast

Greedy descent gets stuck. What now?

Random restarts

Jump to a fresh random state.

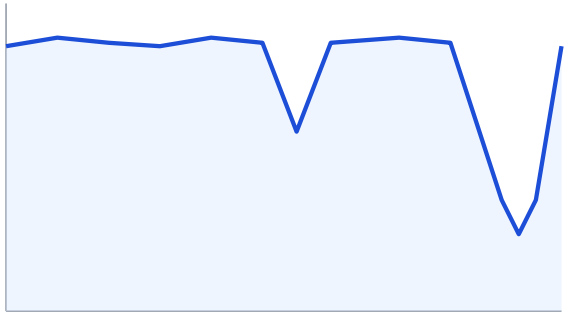
e.g. greedy descent with restarts.

Random walks

Sometimes step to a *random* neighbour, not just a better one.

e.g. simulated annealing.

Q: which random move helps where?



(a) mostly flat, hidden deep valleys

(b) bumpy bowl with one deep valley

Q. Which random move is better for each landscape?

> **(a) Random restarts** — basins are widely separated; a walk can't drift between them, but a fresh restart can land in a different valley.
> **(b) Random walks** — the bumps are small; a few random steps slip out of a shallow local optimum and into the global basin.

Greedy descent with random restarts

> *If at first you don't succeed, try, try again.*

- Run greedy descent multiple times, each from a fresh random state.
- Keep the best state seen across runs.
- **With enough restarts**, finds the global optimum with probability $\to 1$ — eventually a restart will start in the global basin.

Simulated annealing

Annealing: cool a molten metal slowly to make it strong.

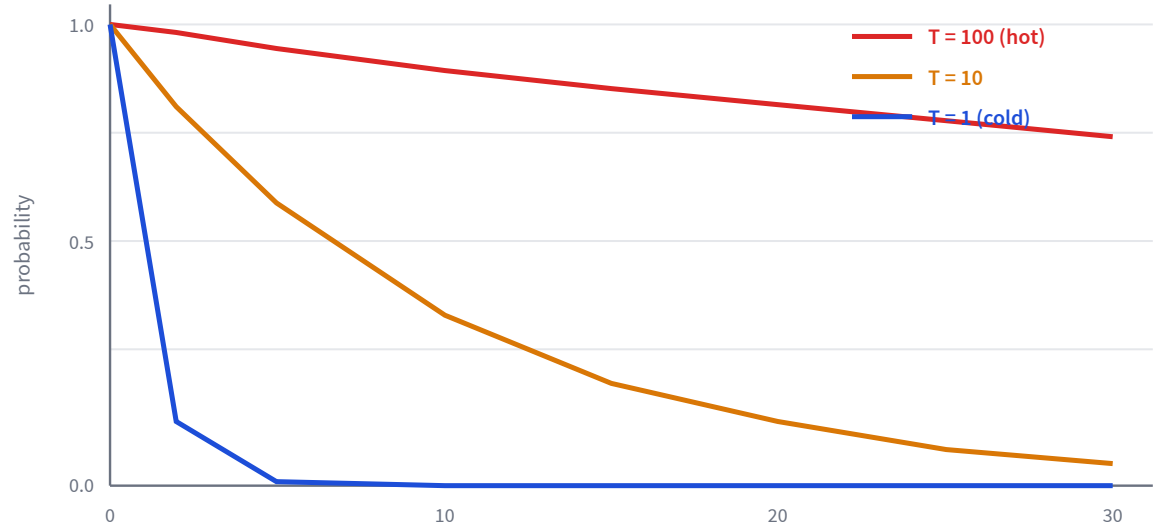
- Start hot — high temperature T; cool over time.
- Each step: try a *random* neighbour.
- Better? Always accept.
- Worse? Accept with probability based on T and how much worse.

High T: exploration. Low T: exploitation.

Probability of accepting a worse neighbour

Current state A, worse neighbour A', $\Delta C = \text{cost}(A') - \text{cost}(A) > 0$, temperature T:

$$p = e^{-\Delta C / T}$$



Hot temperatures stay willing to accept bad moves; cold temperatures barely budge.

Q: as T decreases…

Q. As T *decreases*, the probability of moving to a worse neighbour…

A. increases

B. stays the same

C. decreases

C — decreases. Example with $\Delta C = 10$: $T = 100 \Rightarrow p = e^{-0.1} \approx 0.90$; $T = 10 \Rightarrow p = e^{-1} \approx 0.37$.

Q: as ΔC increases…

Q. As ΔC *increases* (the neighbour is much worse), the probability…

A. increases

B. stays the same

C. decreases

> **C — also decreases.** Example with $T = 10$: $\Delta C = 10 \Rightarrow p \approx 0.37$; $\Delta C = 100 \Rightarrow p \approx 0.000045$.

Simulated annealing algorithm

`simulated-annealing()`

1. `current` ← initial state; `T` ← large positive value.
2. While `T > 0`:
3. Sample a random neighbour `next`.
4. Let $\Delta C = \text{cost}(\text{next}) - \text{cost}(\text{current})$.
5. If $\Delta C < 0$ → accept (`current ← next`); else accept with probability $p = e^{-\Delta C/T}$.
6. Cool T (e.g. $T \leftarrow \alpha\, T$).
7. Return `current`.

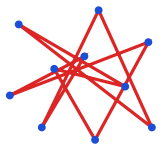
Annealing schedule

How fast should T cool?

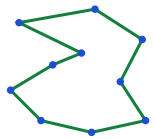
- **In theory:** very slowly. If T decreases slowly enough, simulated annealing finds the global optimum with probability $\to 1$.
- **In practice: geometric cooling** — $T_{k+1} = \alpha T_k$ with α close to 1.

Example: start at $T_0 = 10$, $\alpha = 0.99$. After 500 steps, $T \approx 0.07$.

SA in action: travelling salesman



high T: tangled



low T: smooth

30 cities → $30! \approx 2.6 \times 10^{32}$ tours. NP-hard — yet SA finds great tours fast.

SA as life

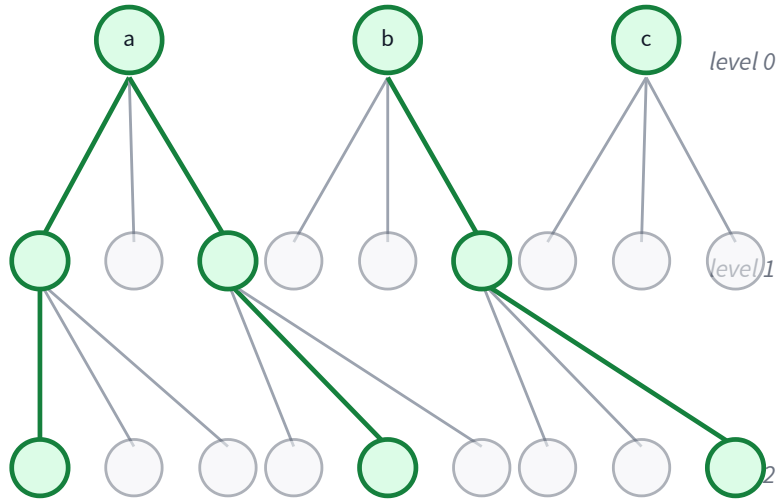
- High T — young, exploring.
- Low T — older, refining.
- Exploration vs. exploitation, all over again.

From a single state to a population

- So far, every algorithm remembers *one* state at a time.
- What if we keep k **states** in parallel?

Beam search

- Keep the k best states in parallel.
- Expand all their neighbours.
- Keep the new k best from the pool.
- k controls memory + parallelism.



Green = the $k = 3$ kept per level.

Beam search — corner cases

$k = 1$	greedy descent
$k = \infty$	breadth-first search
vs. k parallel restarts	beam *shares* across threads; restarts are independent
Watch out	states can clump — lack of diversity

Stochastic beam search

- Pick the next k states **probabilistically**, with probability proportional to fitness (e.g. $\propto e^{-\text{cost}/T}$).
- Maintains **diversity** in the population.
- Mimics natural selection — *asexual* reproduction (mutate, keep best).

Adding *sexual* reproduction gives the **genetic algorithm**.

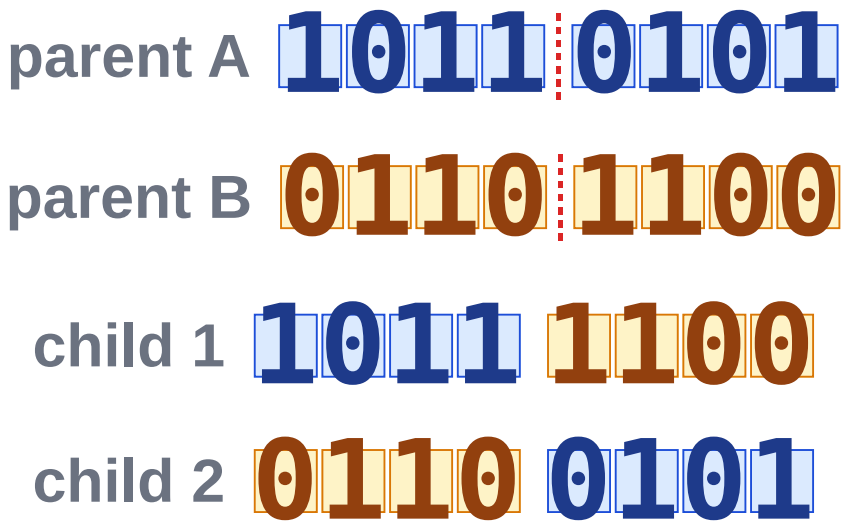
Genetic algorithm

Population evolves; fittest survive; offspring combine and mutate.

`genetic-algorithm()`

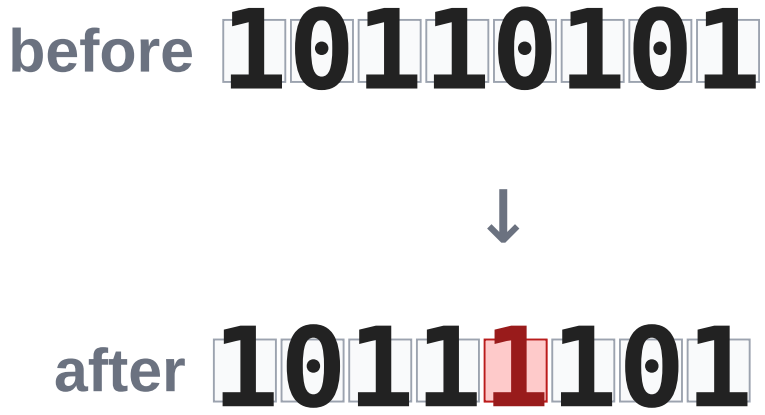
1. Initialise a population P of random individuals.
2. Until converged:
3. Select parents from P weighted by fitness.
4. Crossover parents → offspring.
5. Mutate offspring with small probability.
6. Replace P with the offspring.
7. Return the best individual in P.

Crossover



Split parents at a point; swap tails.

Mutation



Flip a random bit with small probability.

Four local-search algorithms

Algorithm	Explores how	Remembers	Finds global?	Best at
Greedy descent	best neighbour	1 state	**No** *stuck at local optimum*	fast first cut
GD + random restarts	best, then restart	1 state + best so far	**Yes** *in the limit*	landscapes with wide basins
Simulated annealing	random; accept worse with $p = e^{-\Delta C/T}$	1 state + T	**Yes** *slow enough cooling*	bumpy landscapes
Genetic algorithm	crossover + mutation	population of k	*probabilistically*	complex combinatorial spaces

Learning goals (recap)

- ✓ Describe the advantages of local search
- ✓ Formulate a real-world problem as local search
- ✓ Verify whether a state is a local / global optimum
- ✓ Describe strategies for escaping local optima
- ✓ Trace greedy descent, random restarts, simulated annealing, genetic algorithms
- ✓ Compare and contrast local-search algorithms

Search › Uncertainty › Decisions › Learning

Next module: Reasoning under Uncertainty

We've optimized in the *deterministic* world. The agent will now have to reason when it can't be sure of the state or the outcome.

- **L6** — probability rules: sum, product, chain, Bayes.
- **L7–L8** — independence + Bayesian networks + d-separation.
- **L9** — variable elimination for inference.
- **L10–L11** — hidden Markov models, filtering, smoothing, Viterbi.