CPSC 322: Introduction to Artificial Intelligence

CSPs: Stochastic Local Search Variants

Textbook reference: [4.7.3, 4.8]

Instructor: Varada Kolhatkar
University of British Columbia

Credit: These slides are adapted from the slides of the previous offerings of the course. Thanks to all instructors for creating and improving the teaching material and making it available!

Announcements

- Final exam scheduled: **Dec 9 at 7:00pm**
- Reminder: Assignment 2 is due on **Oct 21 at 11:59pm**.
- Reminder: Midterm next Friday, **Oct 25 from 6pm to 7pm** in Woodward 2
- Midterm **practice questions** are available on Piazza.
- I will be holding **extra office hours** next week on Thursday during class time (5 to 6:30pm) in ICCS 185.

Midterm

- Included in midterm: Everything till today's lecture
- Planning won't be included in midterm
- Tips on studying for midterm
  - Make sure you understand the purpose and basic ideas of the different algorithms we have learned so far.
  - Go through lecture notes and the lecture learning outcomes and make sure you can do things that are expected of you
  - Go through the iclicker questions to test your understanding
  - Make use of extra office hours next week

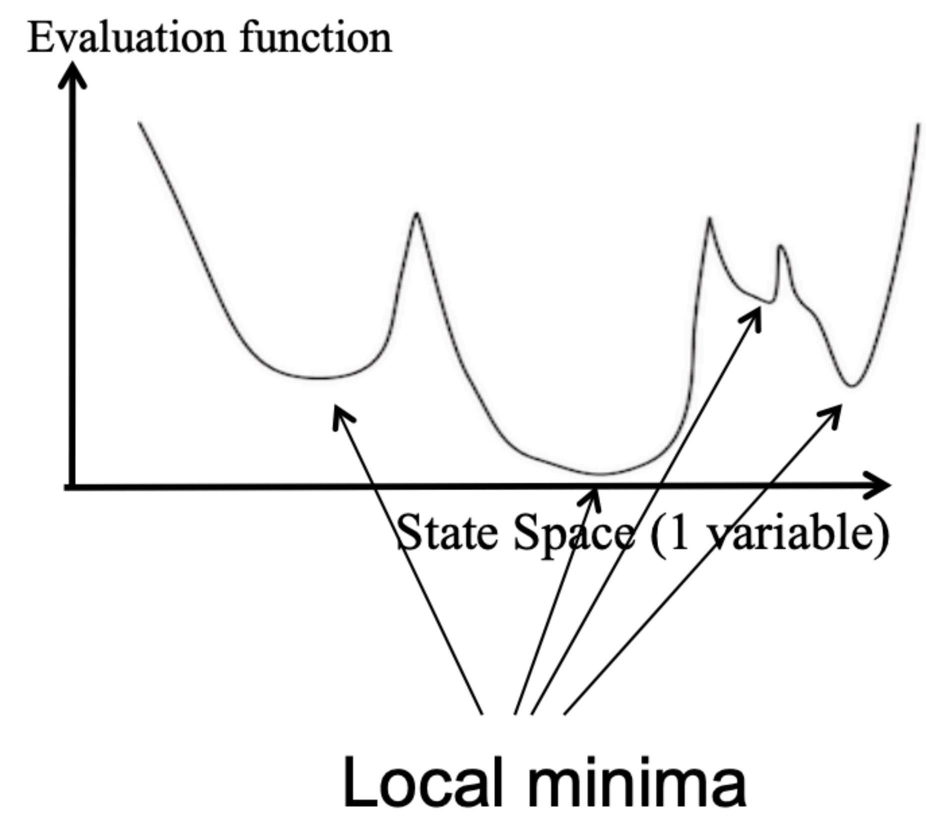
Lecture outline

- Recap stochastic local search (~5 mins) 👈
- SLS variants (~50 mins)
  - Tabu lists
  - Simulated Annealing
  - Beam search
  - Genetic algorithms
- Summary and wrap up (~5 mins)

Local search problem: Local minima

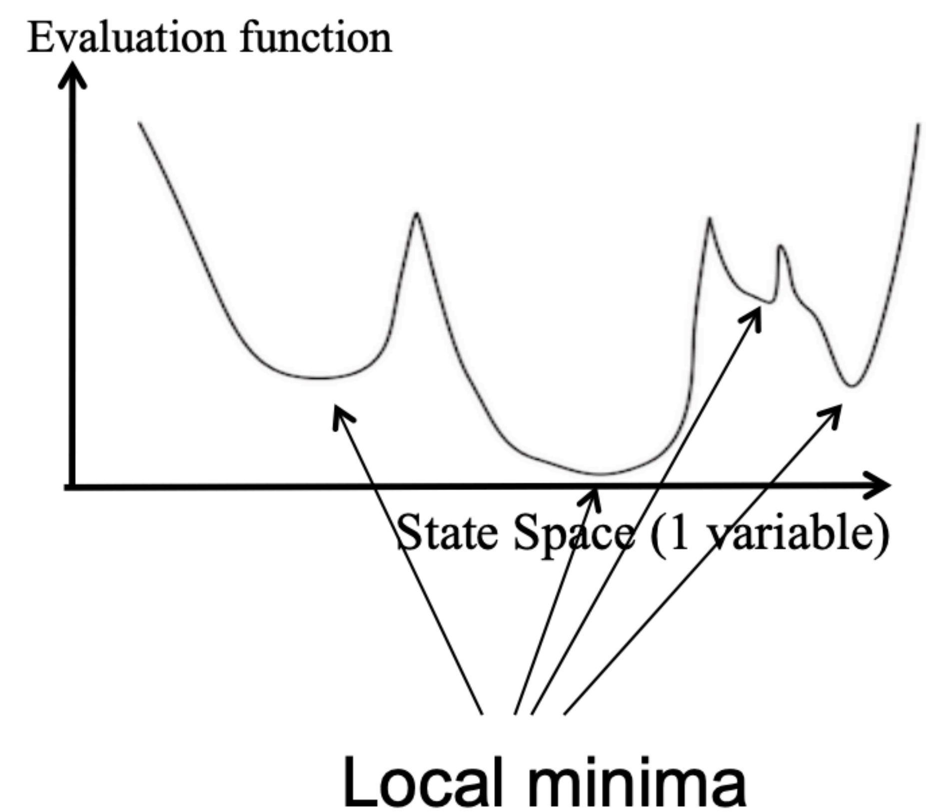
The primary problem associated with local search methods (i.e., greedy descent and hill climbing) is getting stuck in local minima/maxima.

The greedy methods always choose a "better" scoring neighbour and there is no such neighbour at local minima/maxima.



Local search problem: Local minima

The primary problem associated with local search methods (i.e., greedy descent and hill climbing) is getting stuck in local minima/maxima.



The greedy methods always choose a "better" scoring neighbour and there is no such neighbour at local minima/ maxima.

Solution: Stochastic local search!! Add randomness to avoid getting trapped in local minima! 💡

Greedy descent vs. Random sampling

- **Greedy descent** is good for finding local minima – bad for exploring new parts of the search space

- **Random sampling** is good for exploring new parts of the search space – bad for finding local minima

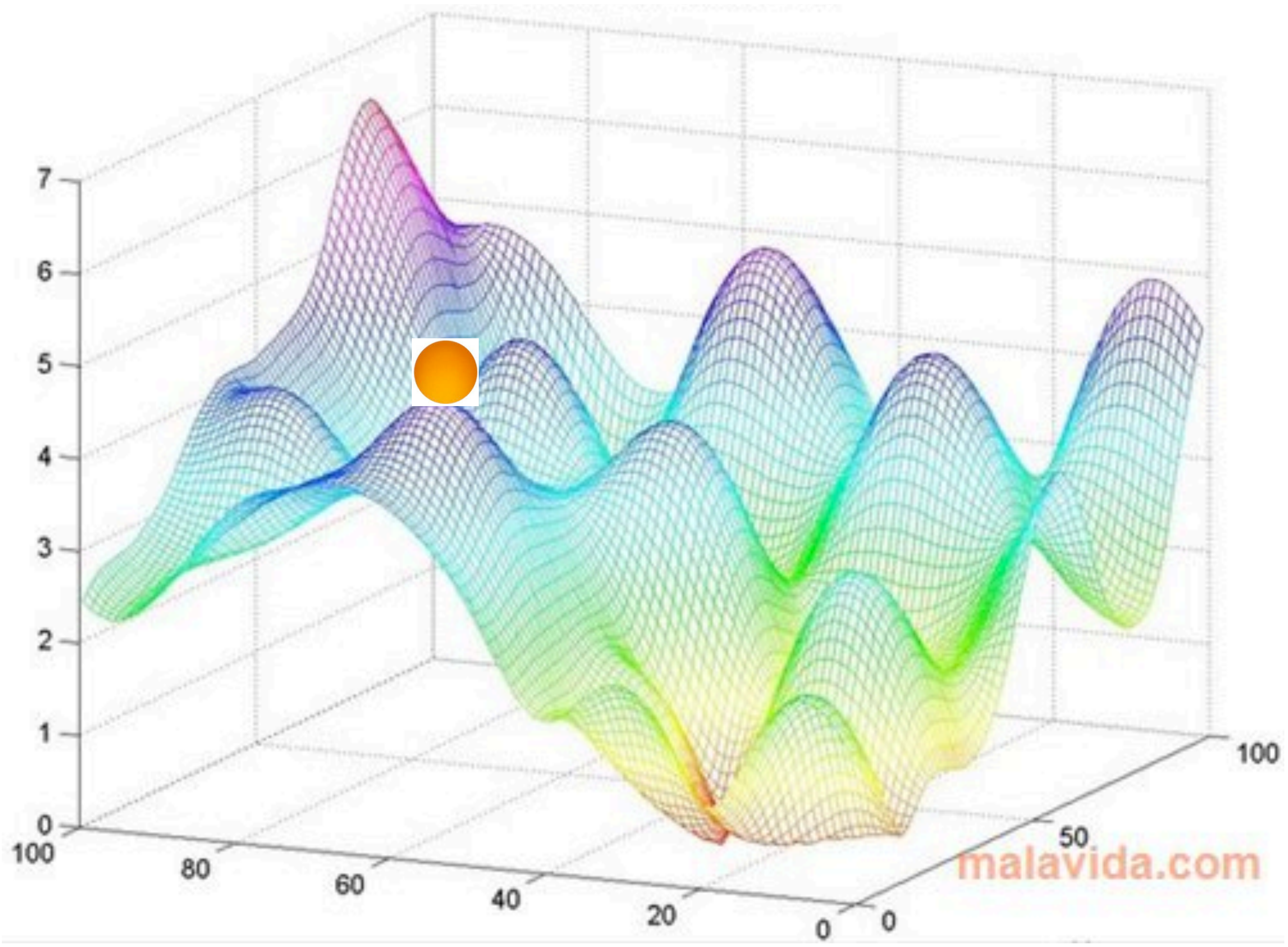
A mix of the two can work very well.

Adding randomness to greedy descent

- **Random restart**: reassign random values to all variables (i.e. start fresh)

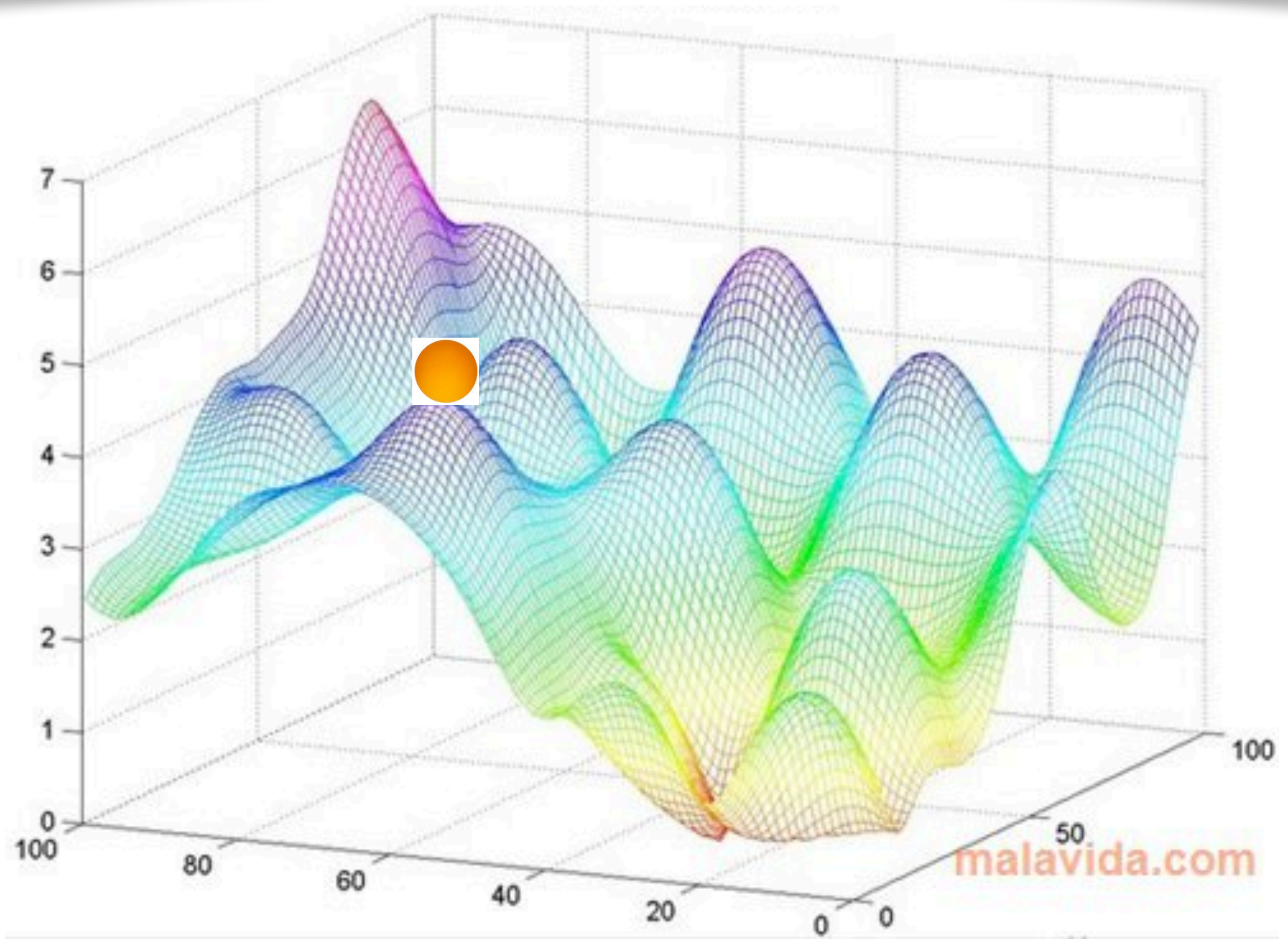
- **Random step**: move to a random neighbour

Adding randomness



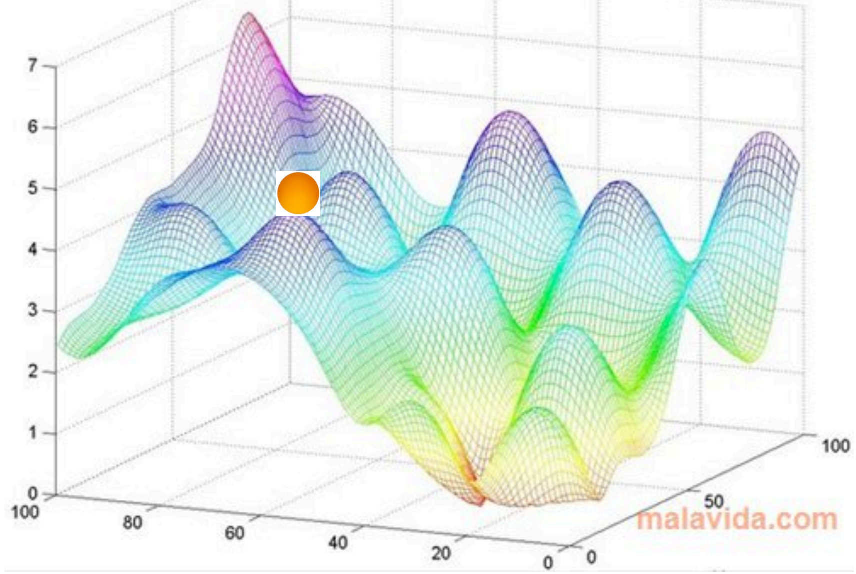
Adding randomness

Consider the task of getting a ping-pong ball into the deepest crevice



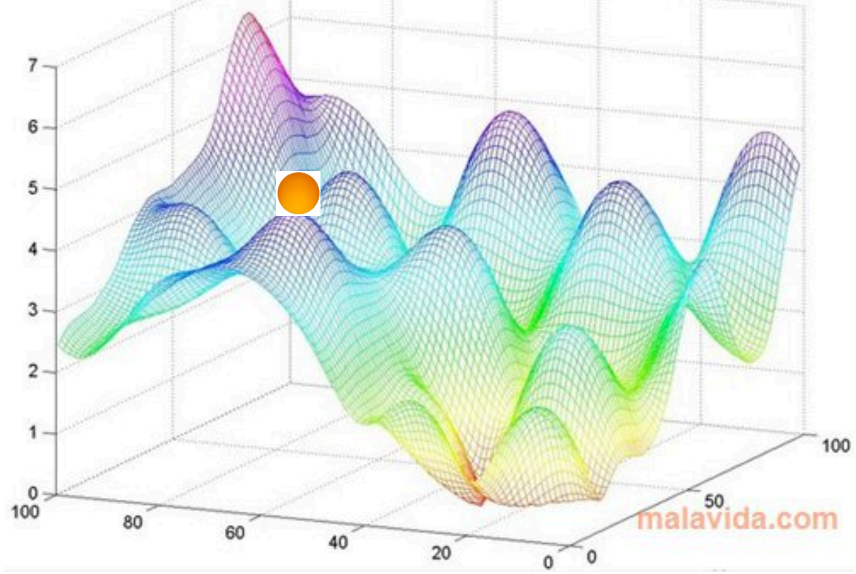
Adding randomness

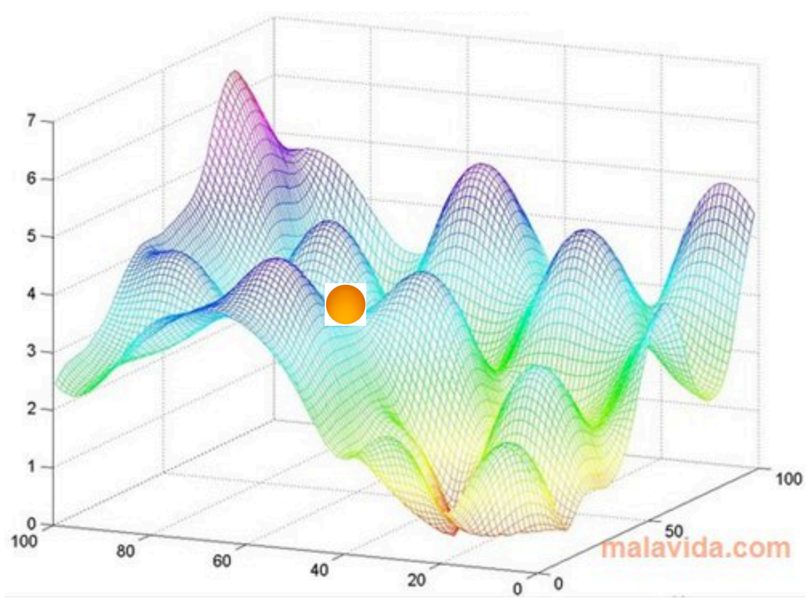
Current state



Adding randomness

Current state

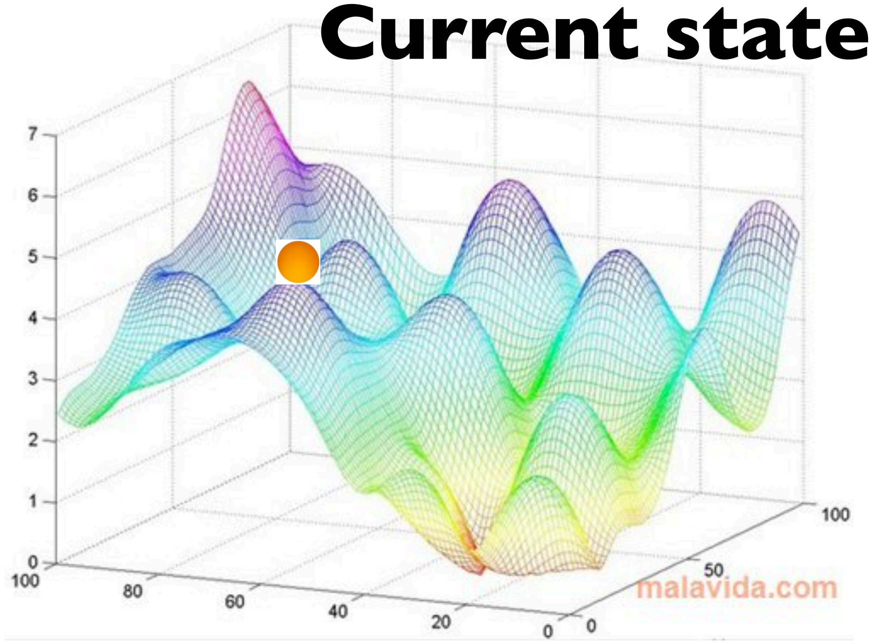


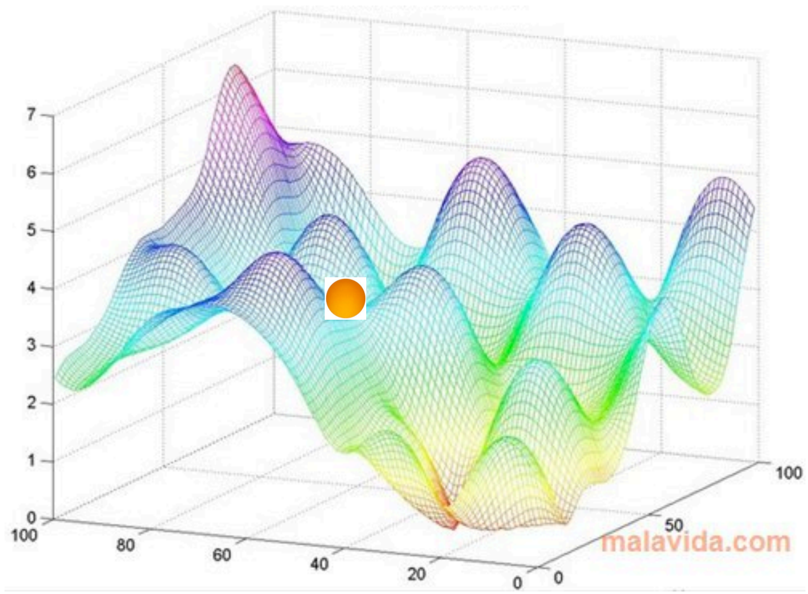


Greedy descent: Let it roll.
Move to the “best” neighbour.

Adding randomness

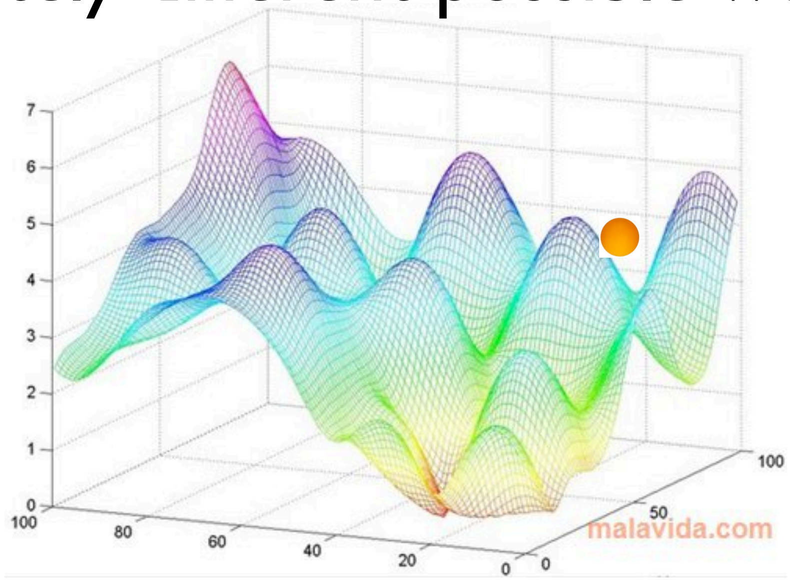
Current state





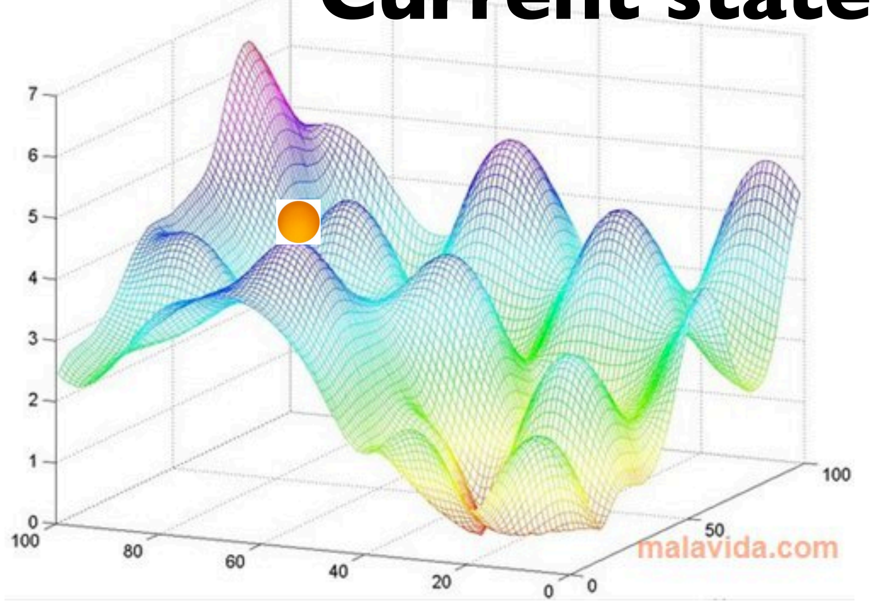
Greedy descent: Let it roll.
Move to the "best" neighbour.

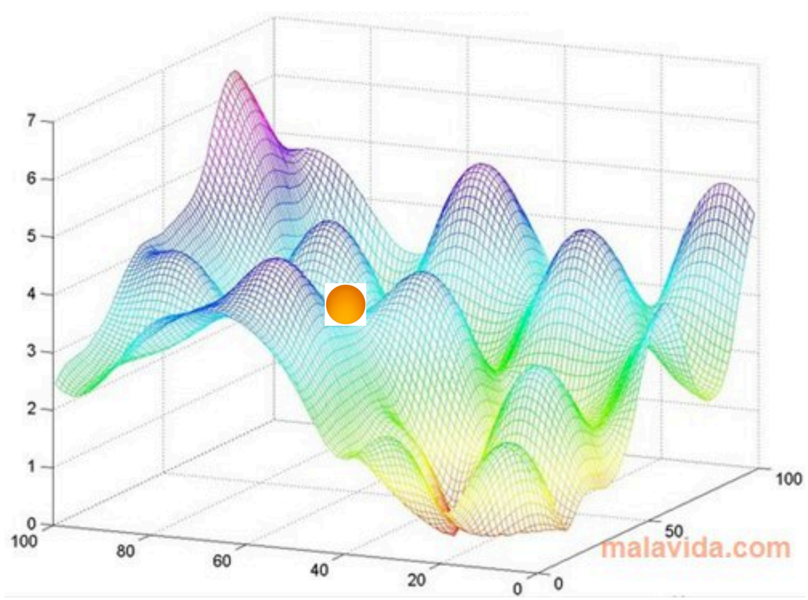
Random restart: Move to a completely different possible world



Adding randomness

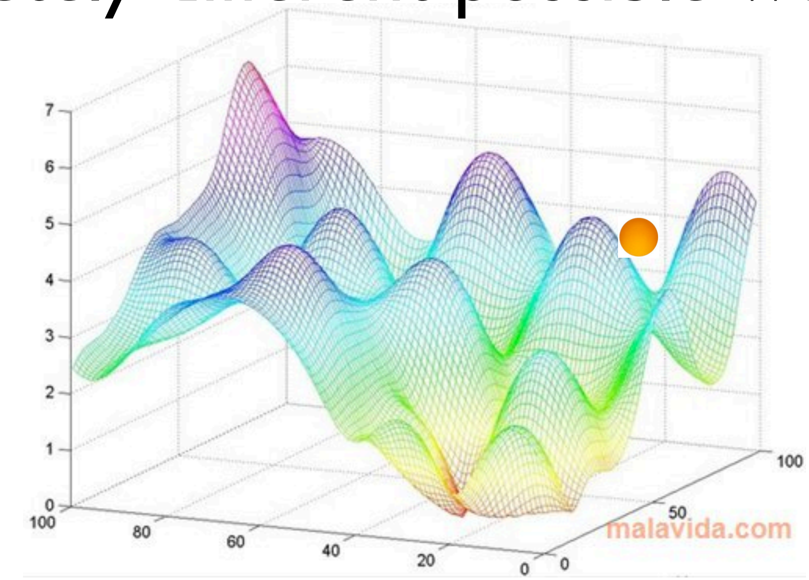
Current state



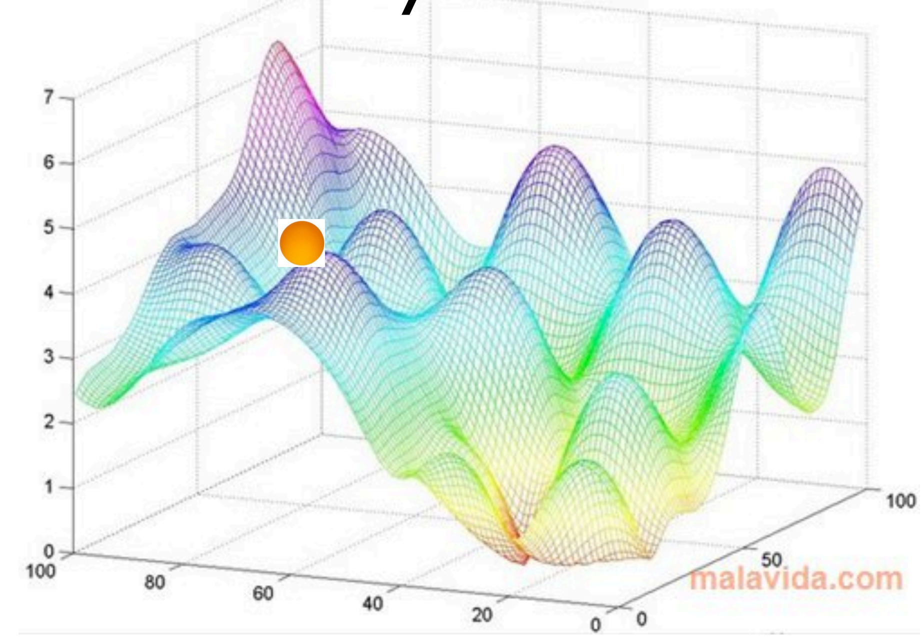


Greedy descent: Let it roll. Move to the “best” neighbour.

Random restart: Move to a completely different possible world



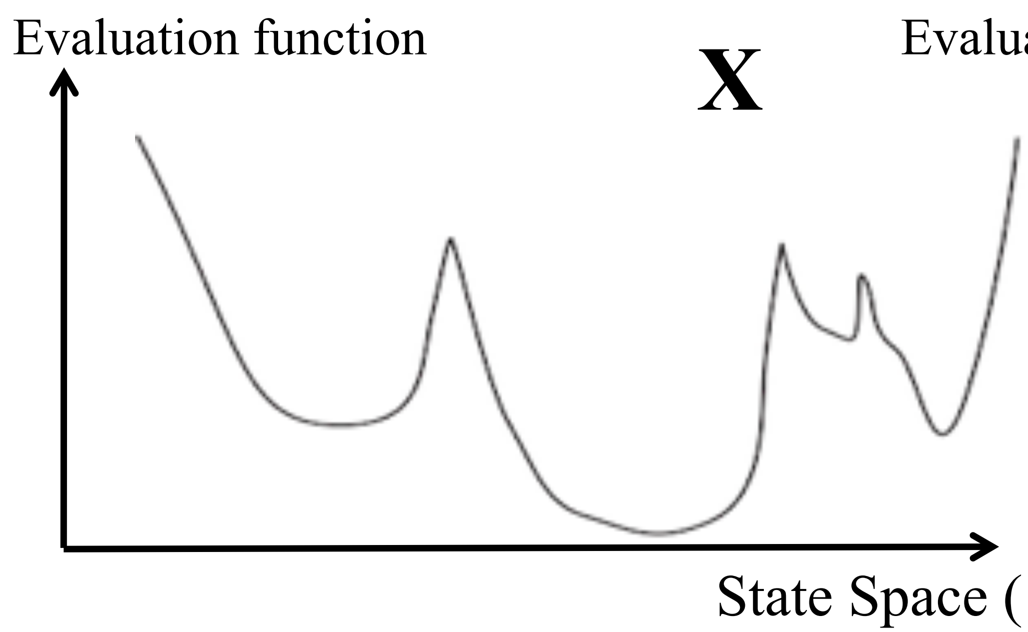
Random step: Move to a **random** neighbour, not necessarily the “best” one

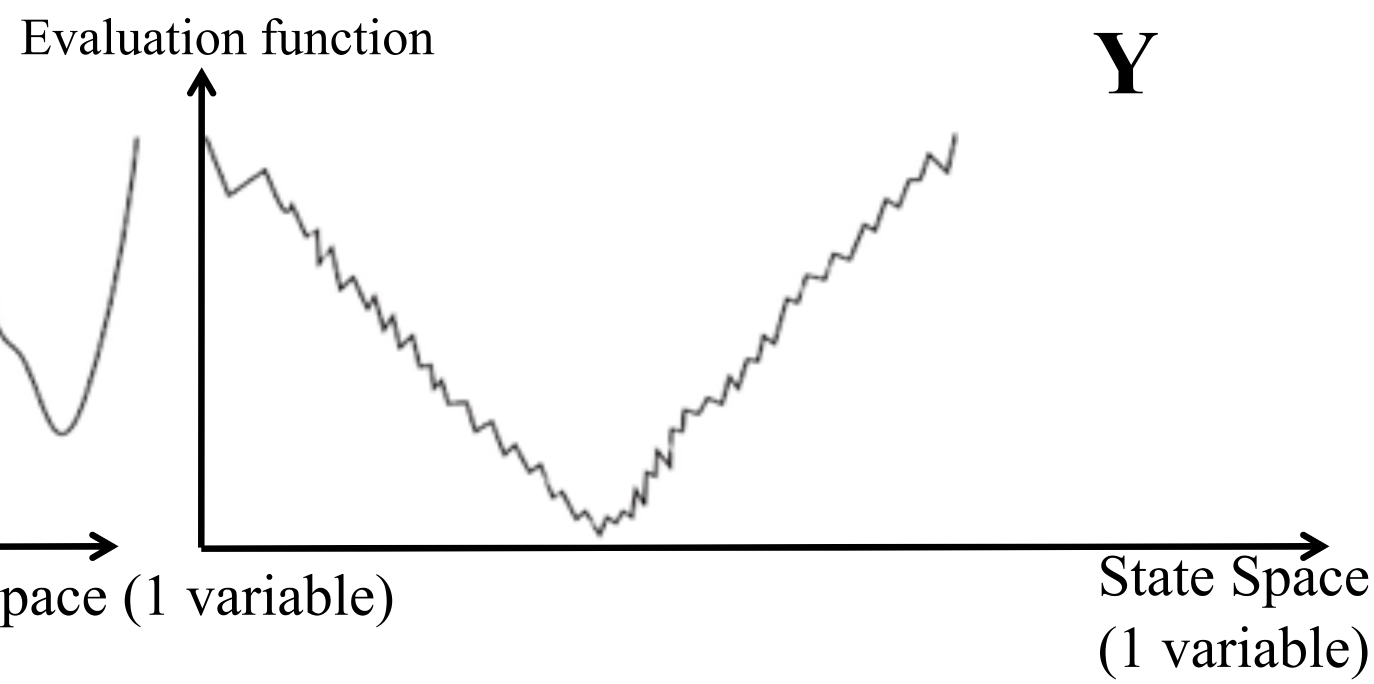


Greedy descent + randomness

Greedy descent with random restart for X

Greedy descent with random steps for Y





Today: Learning outcomes

From this lecture, students are expected to be able to:

- Explain the principles and ideas behind the following SLS variants
- Tabu list
- Simulated annealing
- Beam search
- Genetic algorithms

SLS variants

- There are many different SLS algorithms
- Each could easily be a lecture by itself
- We will only touch on each of them very briefly
- If you want to know more:
  - See this book "Stochastic Local Search: Foundations and Applications" by Holger Hoos & Thomas Stützle, 2004 (available in reading room)

Lecture outline

- Recap stochastic local search (~5 mins)
- SLS variants (~50 mins)
  - Tabu lists 👈
  - Simulated Annealing
  - Beam search
  - Genetic algorithms
- Summary and wrap up (~5 mins)

Tabu list idea

Greedy descent + randomness + memory

- **Randomness:** accepts a worse neighbour if no better neighbour is available
- **Memory:** Avoid potentially problematic behaviour by maintaining a tabu list.
  - E.g., Returning to recently visited nodes or cycling

Tabu list

- Mark partial assignments as tabu ('taboo'= forbidden)
- Prevents repeatedly visiting the same (or similar) local minima
- Maintain a queue of k variable = value assignments that are tabu
  - E.g., when changing V7 's value from 2 to 4, we cannot change V7 back to 2 for the next k steps
  - k is a parameter that needs to be optimized empirically

Lecture outline

- Recap stochastic local search (~5 mins)
- SLS variants (~50 mins)
  - Tabu lists
  - Simulated Annealing 👈
  - Beam search
  - Genetic algorithms
- Summary and wrap up (~5 mins)

Simulated annealing

Key idea: Change the degree of randomness. Escape local minima by initially allowing some “bad” moves. Then gradually decrease the frequency of such moves.

Annealing: a metallurgical process where metals are heated and then slowly cooled.

- Analogy: start with a high “temperature”: a high tendency to take random steps

- Over time, cool down: more likely to follow the scoring function (only take random steps that are not too bad)

Temperature reduces over time, according to an annealing schedule

Simulated annealing: algorithm

Here's how it works (for maximizing $h(n)$):

- You are in node n. Pick a variable at random and a new value for it at random. You generate a neighbour node n'.

- If it **is** an "improvement" i.e., $h(n') >= h(n)$, adopt it.

- If it **isn't** an improvement, adopt it **probabilistically** depending on the difference and the temperature T.

  - We move to n' with probability: $e^{\frac{h(n') - h(n)}{T}}$

Pair-share (2 mins)

If it **isn't** an improvement, adopt it probabilistically depending on the difference and the temperature T.

- We move to n' with probability: $e^{\frac{h(n') - h(n)}{T}}$

Discuss with your neighbour the effect of higher and lower values of T on the probability of moving to n'.

Simulated annealing: "temperature" parameter

iclicker.

If it **isn't** an improvement, we move to n' with probability: $e^{\frac{h(n') - h(n)}{T}}$

Having a higher temperature T means that the algorithm is

A. more likely to move to n' if it is "better" than n

B. less likely to move to n' if it is "better" than n

C. more likely to move to n' if it is "worse" than n

D. less likely to move to n' if it is "worse" than n

Simulated annealing: "temperature" parameter

i▸clicker

If it **isn't** an improvement, we move to n' with probability: $e^{\frac{h(n') - h(n)}{T}}$

Having a higher temperature T means that the algorithm is

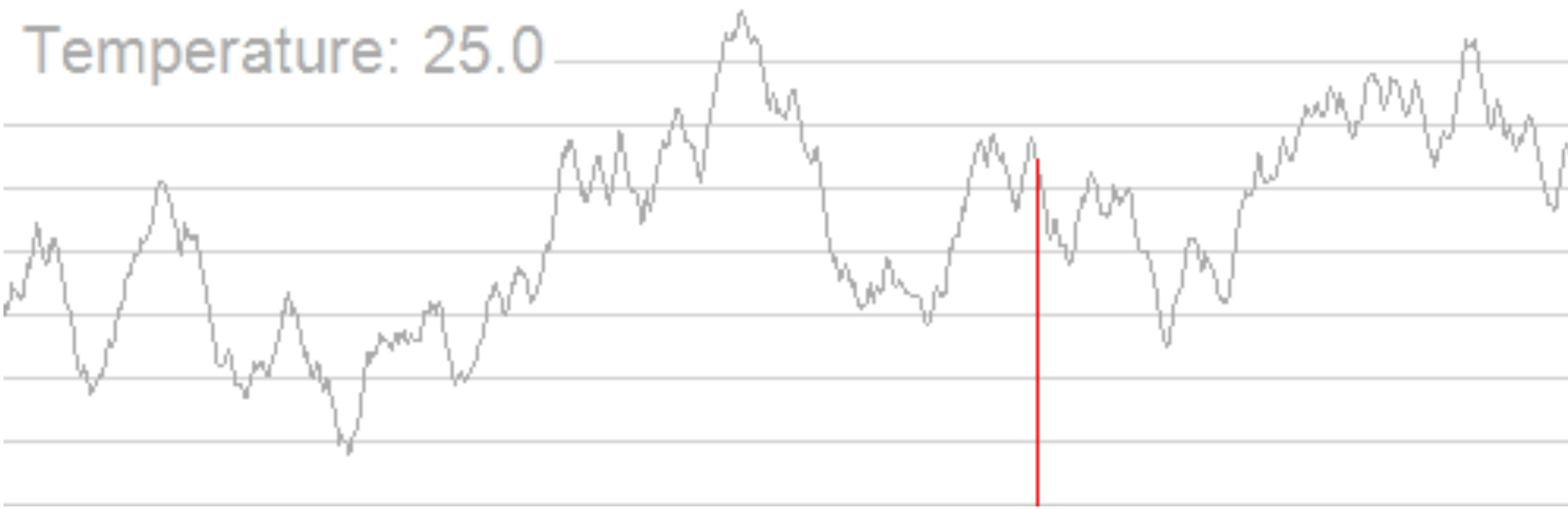
A. more likely to move to n' if it is "better" than n

B. less likely to move to n' if it is "better" than n

C. more likely to move to n' if it is "worse" than n ✅

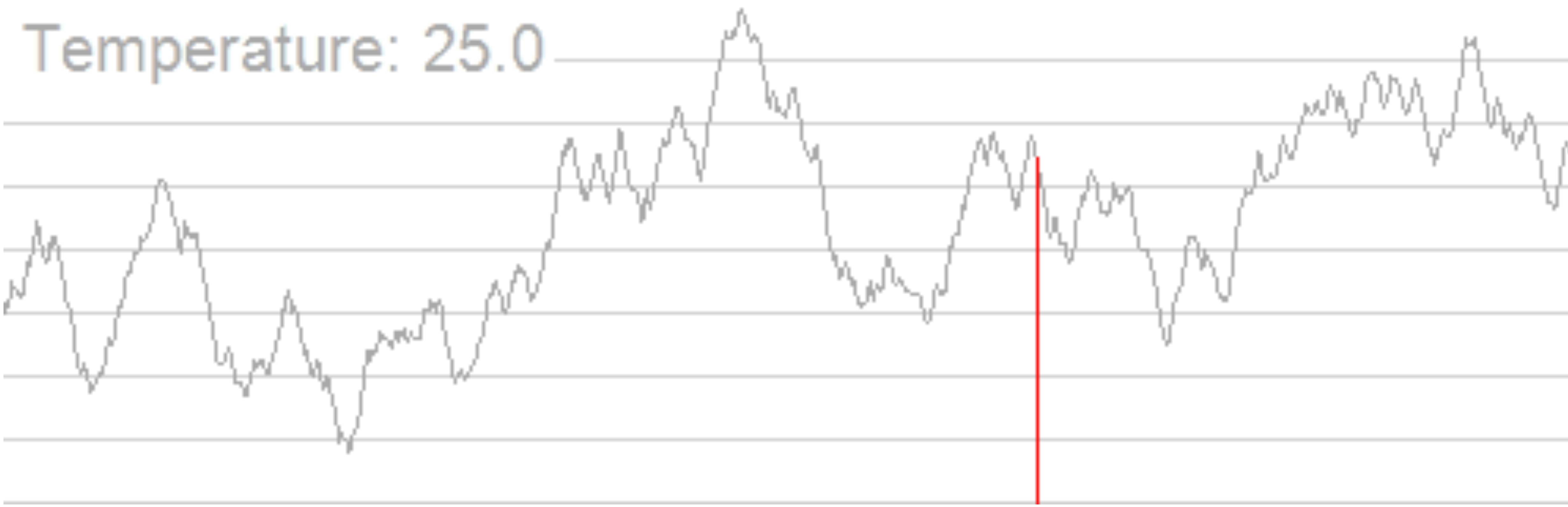
D. less likely to move to n' if it is "worse" than n

Simulated annealing: T parameter



Source: https://en.wikipedia.org/wiki/Simulated_annealing

Simulated annealing: T parameter



Source: https://en.wikipedia.org/wiki/Simulated_annealing

Simulated annealing: temperature parameter

If it **isn't** an improvement, we move to n' with probability: $e^{\frac{h(n') - h(n)}{T}}$

Higher T → higher probability for given $h(n') - h(n)$

Higher $|h(n') - h(n)|$ → smaller probability for given T

Properties of simulated annealing

- One can prove: If T decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1.
- Widely used in VLSI layout, airline scheduling, etc.
- Finding the ideal cooling schedule is unique to each class of problems
  - Want to move off of local maxima but not global maxima
  - Want to minimize computation time while taking enough time to ensure we reach a good minimum

Lecture outline

- Recap stochastic local search (~5 mins)
- SLS variants (~50 mins)
  - Tabu lists
  - Simulated Annealing
  - Beam search 👈
  - Genetic algorithms
- Summary and wrap up (~5 mins)

Population based SLS

Key Idea: maintain a population of k individuals

- Often we have more memory than the one required for current node (+ best so far + tabu list)
- Maintain a population of k individuals
- At every stage, update your population.
- Whenever one individual is a solution, report it.

Beam search

- Keep not just 1 assignment, but k assignments at once
- A 'beam' with k different assignments (k is the 'beam width')
- Useful information is passed among the k parallel search threads
- The neighbourhood is the union of the k neighbourhoods
- At each step, keep only the k best neighbours
- Never backtrack

The value of k lets us limit space and parallelism.

Beam search

Keep not just 1 assignment,
but k assignments at once

When k = 1, beam search is identical to

A. Best-first search

B. Greedy descent

C. Breadth-first search

Beam search

Keep not just 1 assignment,
but k assignments at once

When k = 1, beam search is identical to

A. Best-first search

B. Greedy descent ✅

C. Breadth-first search

Beam search

Keep not just 1 assignment,
but k assignments at once

When $k = \infty$, beam search is identical to

A. Best-first search

B. Greedy descent

C. Breadth-first search

Beam search

Keep not just 1 assignment,
but k assignments at once

When $k = \infty$, beam search is identical to

A. Best-first search

B. Greedy descent

C. Breadth-first search ✅

Beam search

Keep not just 1 assignment,
but k assignments at once

When $k = \infty$, beam search is identical to

A. Best-first search

B. Greedy descent

C. Breadth-first search

At step m, the beam contains all nodes m steps away from the start node, like breadth-first search, but expanding a whole level of the search tree at once

Beam search: Problems

- Troublesome case: If one individual generates several good neighbours and the other $k-1$ all generate bad successors…
  - Lack of diversity
  - The next generation will comprise very similar individuals

Stochastic beam search

Non-stochastic Beam Search may suffer from lack of diversity among the k individual (just a more expensive hill climbing)

Stochastic version alleviates this problem

- Selects k individuals at random
- But probability of selection proportional to their value (according to scoring function)

Stochastic beam search

Stochastic beam search selects the k individual at random but probability of selection is proportional to their value (according to scoring function).

Suppose node n has m neighbours: $\{n_1, n_2, \ldots, n_m\}$
Let the scoring function be h. Then the probability of selecting n_j is

A. $\dfrac{n_j}{\sum_i n_i}$

B. $\dfrac{h(n_j)}{\sum_i h(n_i)}$

C. $\dfrac{\sum_i h(n_i)}{h(n_j)}$

Stochastic beam search

Stochastic beam search selects the k individual at random but probability of selection is proportional to their value (according to scoring function).

Suppose node n has m neighbours: $\{n_1, n_2, \ldots, n_m\}$
Let the scoring function be h. Then the probability of selecting n_j is

A. $\dfrac{n_j}{\sum_i n_i}$

✅ B. $\dfrac{h(n_j)}{\sum_i h(n_i)}$

C. $\dfrac{\sum_i h(n_i)}{h(n_j)}$

Stochastic beam search: advantages

- It maintains diversity in the population.
- Biological metapor (asexual reproduction):
  - Each individual generates "mutated" copies of itself (its neighbours)
  - The scoring function value reflects the fitness of the individual. The higher the fitness the more likely the individual will survive (i.e., the neighbour will be in the next generation)

Lecture outline

- Recap stochastic local search (~5 mins)
- SLS variants (~50 mins)
  - Tabu lists
  - Simulated Annealing
  - Beam search
  - Genetic algorithms 👈
- Summary and wrap up (~5 mins)

Genetic algorithms

Key Idea: Like stochastic beam search, but pairs of nodes are combined to create the offspring.

Genetic algorithms is a large research field

- Motivated by theory of evolution
- Appealing biological metaphor
- Several conferences are devoted to the topic

Genetic algorithms

- Start with k randomly generated individuals (population)
- An individual is represented as a string over a finite alphabet (often a string of 0s and 1s)
- A successor is generated by combining two parent individuals (loosely analogous to how DNA is spliced in sexual reproduction)
- Evaluation/Scoring function (fitness function): Higher values for better individuals.
- Produce the next generation of individuals by **selection**, **crossover**, and **mutation**

Genetic algorithms

Like stochastic beam search, but pairs of nodes are combined to create the offspring. For each generation:

- Choose pairs of nodes n_1 and n_2 ('parents'), where nodes with low $h(n)$ are more likely to be chosen from the population
- For each pair (n_1, n_2), perform a cross-over: create offspring combining parts of their parents
- Mutate some values for each offspring.
- Select from previous population and all offspring which nodes to keep in the population

Genetic algorithms: Crossover example

- Given two nodes with 8-digit representation:
  $n_1 = 13573333$ and $n_2 = 24682222$

- Select i at random and form two offspring:

- For example, for $i = 4$, what would be the offspring?

Genetic algorithms: Crossover example

- Given two nodes with 8-digit representation:
  $n_1 = 13573333$ and $n_2 = 24682222$

- Select i at random and form two offspring:

- For example, for $i = 4$, the two offsprings would be
  $n_3 =$ 13572222 and $n_4 =$ 24683333

Genetic algorithms: Crossover

Given two nodes:

$X_1 = a_1, X_2 = a_2, \ldots, X_m = a_m$ and

$X_1 = b_1, X_2 = b_2, \ldots, X_m = b_m$

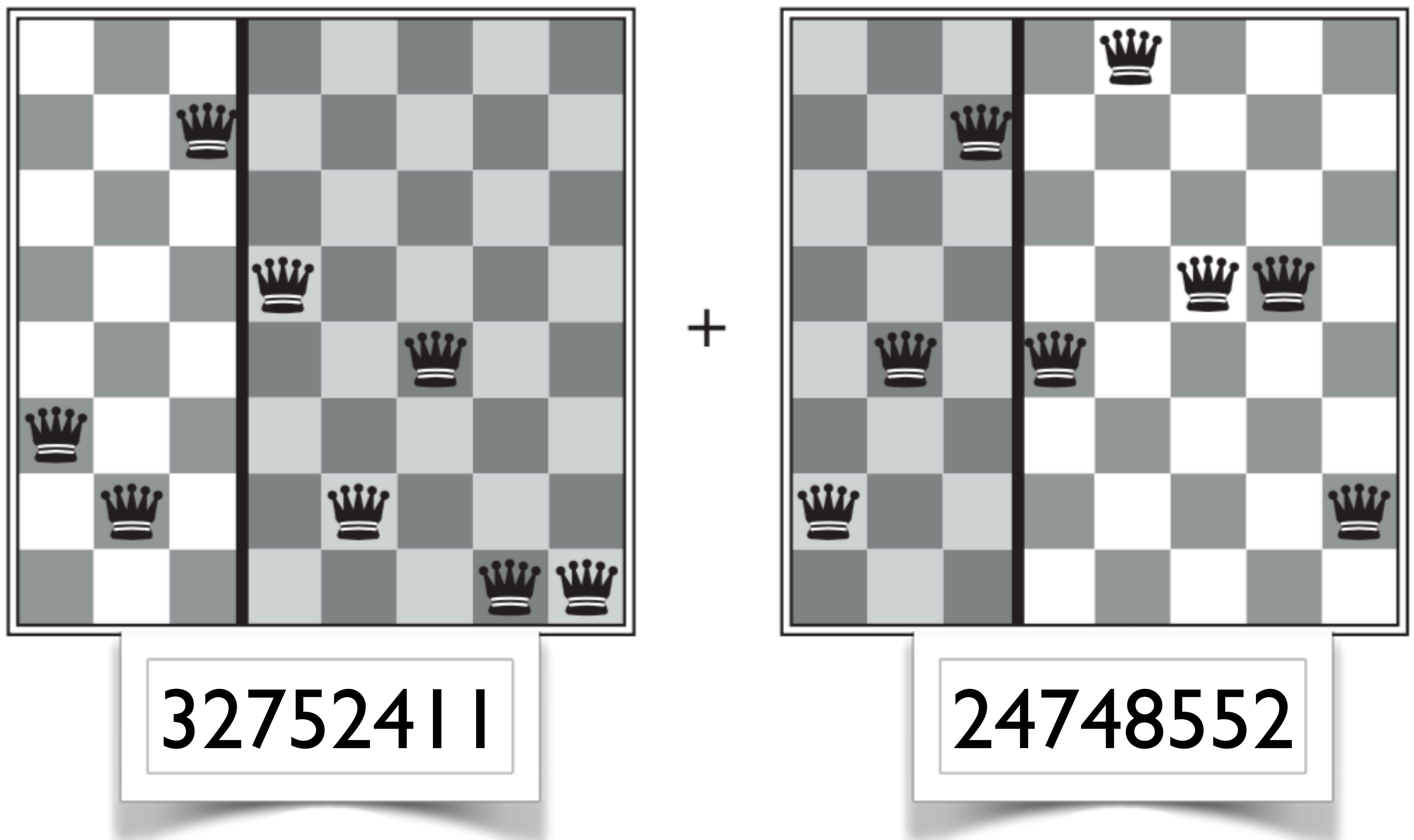
Select i at random and form two offspring:

$X_1 = a_1, \ldots, X_i = a_i, X_{i+1} = b_{i+1}, \ldots, X_m = b_m$ and

$X_1 = b_1, \ldots, X_i = b_i, X_{i+1} = a_{i+1}, \ldots, X_m = a_m$

Many different crossover operators are possible.

Genetic algorithms: Example 8-Queen



State: String over finite alphabet (8-digit representation)

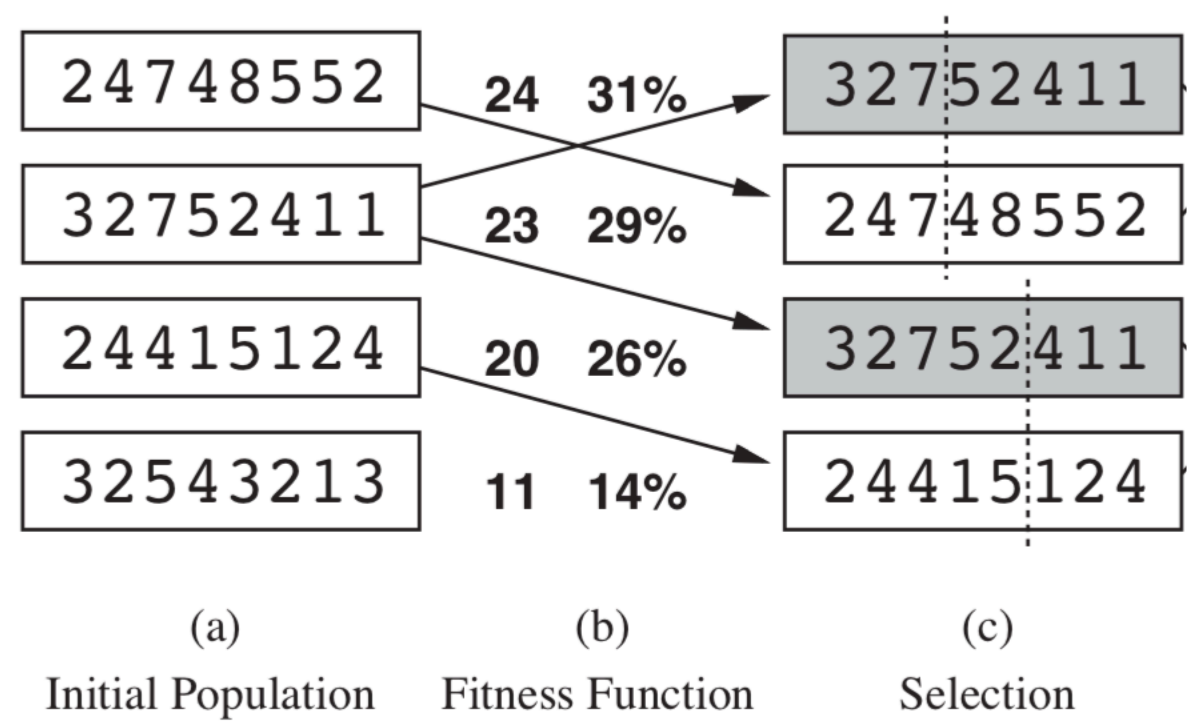
Fitness function: higher values means better states

Genetic algorithms: Example 8-Queen

Selection: common strategy, probability of being chosen for reproduction is directly proportional to fitness score (as with beam search)

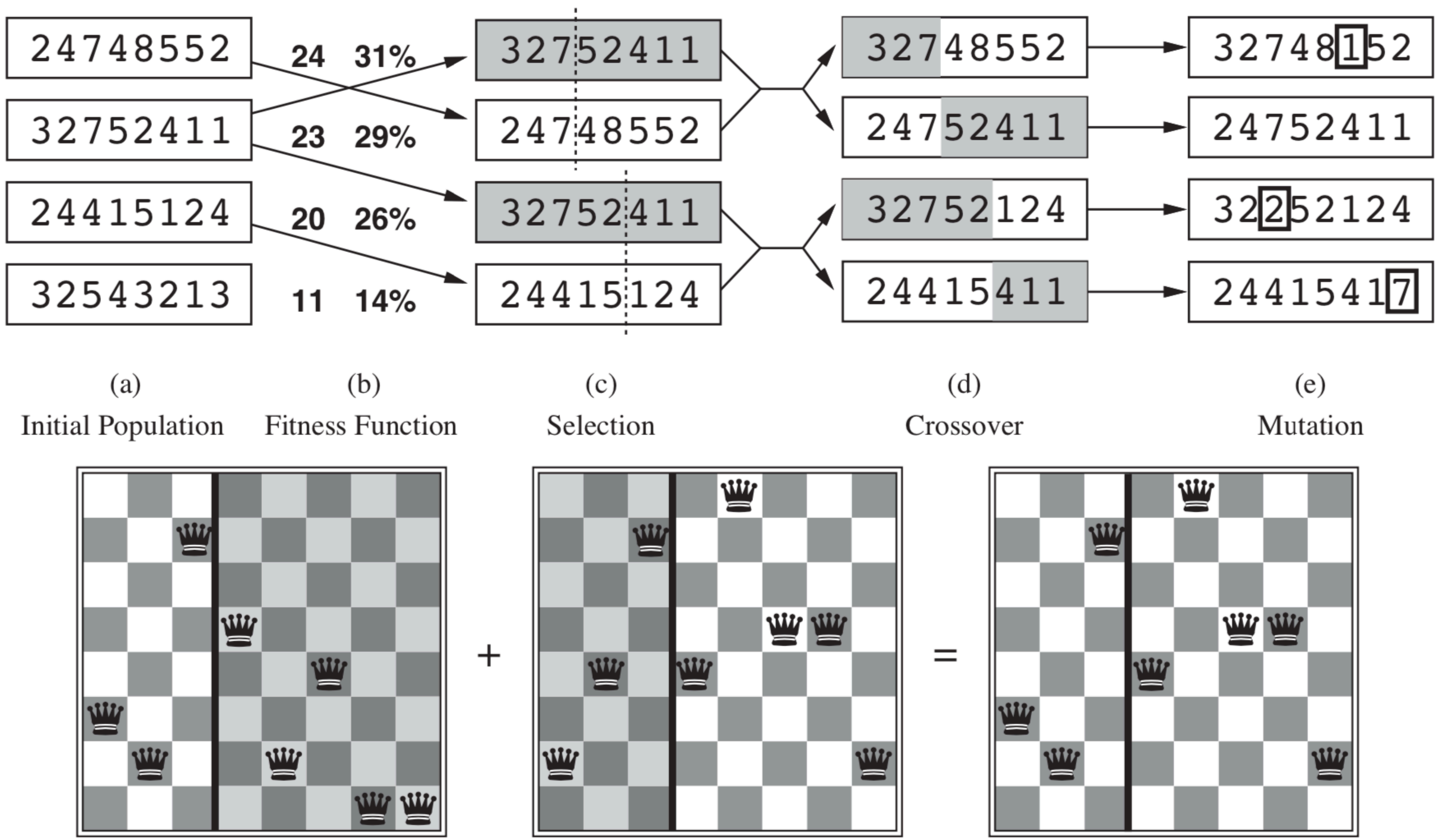
Fitness function: Number of non-attacking queens. Higher value means better state.

$$24/(24 + 23 + 20 + 11) = 31\%$$
$$23/(24 + 23 + 20 + 11) = 29\%$$



Genetic algorithms: Example 8-Queen

Reproduction: Cross-over and mutation



Genetic algorithms: Conclusion

- Their performance is very sensitive to the choice of state representation and fitness function
- Extremely slow (not surprising as they are inspired by evolution!)
- Ongoing work to make them practical for real-world problems

Parameters in SLS

Algorithm	**Parameters**
Simple SLS	Neighbourhoods, variable and value selection heuristics, percentages of random steps, restart probability
Tabu search	Tabu length (or interval for randomized tabu length)
Simulated annealing	Temperature
Beam search	*k* (beam width)
Genetic algorithms	Population size, mating scheme, cross-over operator, mutation rate

Revisit: learning outcomes for local search

- Implement local search for a CSP.
- Implement different ways to generate neighbours
- Implement scoring functions to solve a CSP by local search through either greedy descent or hill-climbing.
- Implement SLS with
    - random steps (1-step, 2-step versions) and random restart
- Compare SLS algorithms with runtime distributions
- Understand principles of types of SLS algorithms

A rough CPSC 322 overview

Representation and reasoning

Environment

Problem		Deterministic	Stochastic
Static	Constraint satisfaction	Arc consistency Variables + constraints Search	
	Query	Logics Search	Belief networks Variable elimination
Sequential	Planning	<u>STRIPS</u> Search	Decision networks Variable elimination Markov decision processes Value iteration

Coming up

Planning: How to select and organize a sequence of actions to achieve a given goal…

Readings for next class

- 6.1 Representing States, Actions, and Goals
- 6.2 Forward Planning
- 6.4 Planning as a CSP