Chapter 4

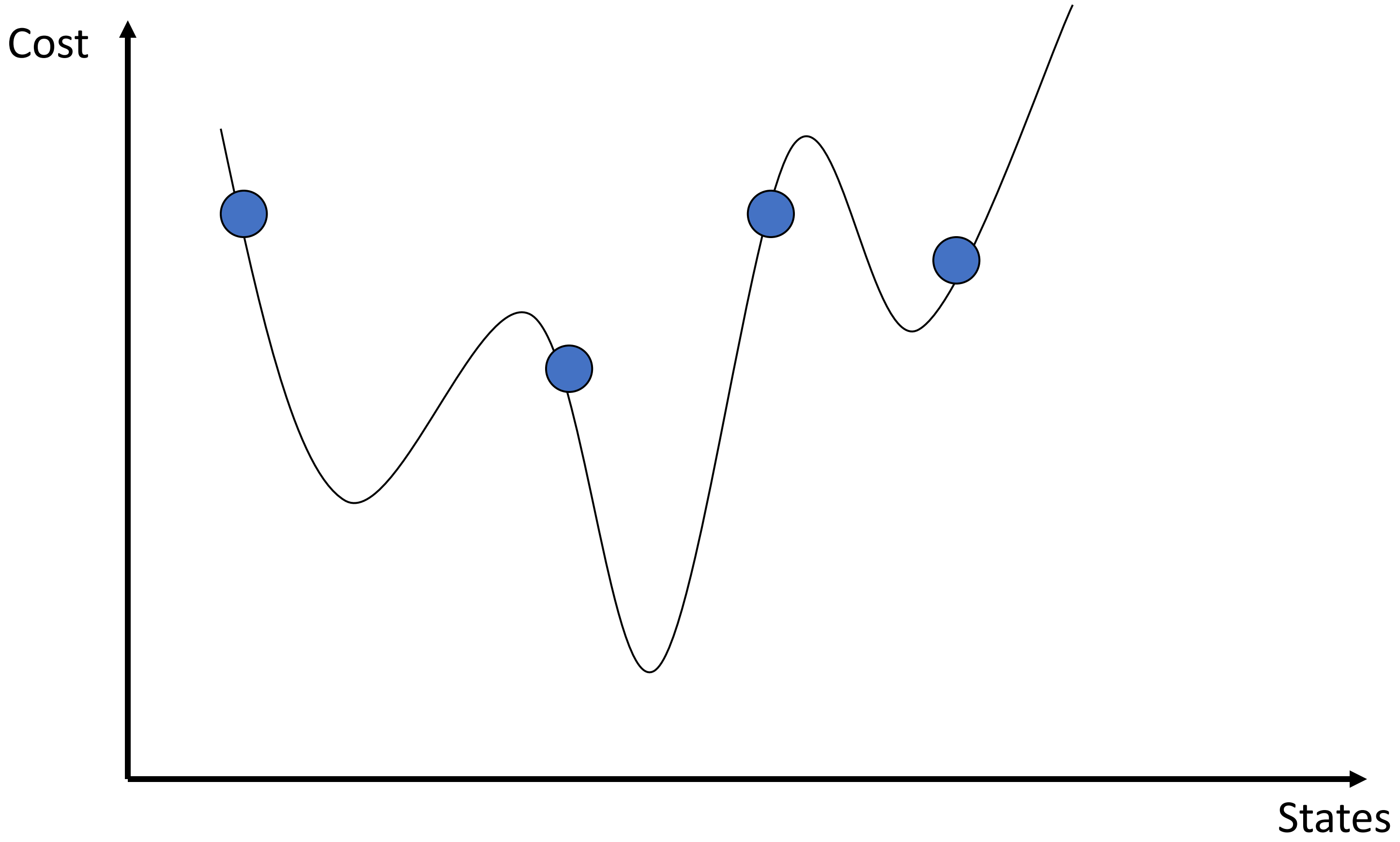
- Beyond Classical Search: Local Search

3. Local beam search

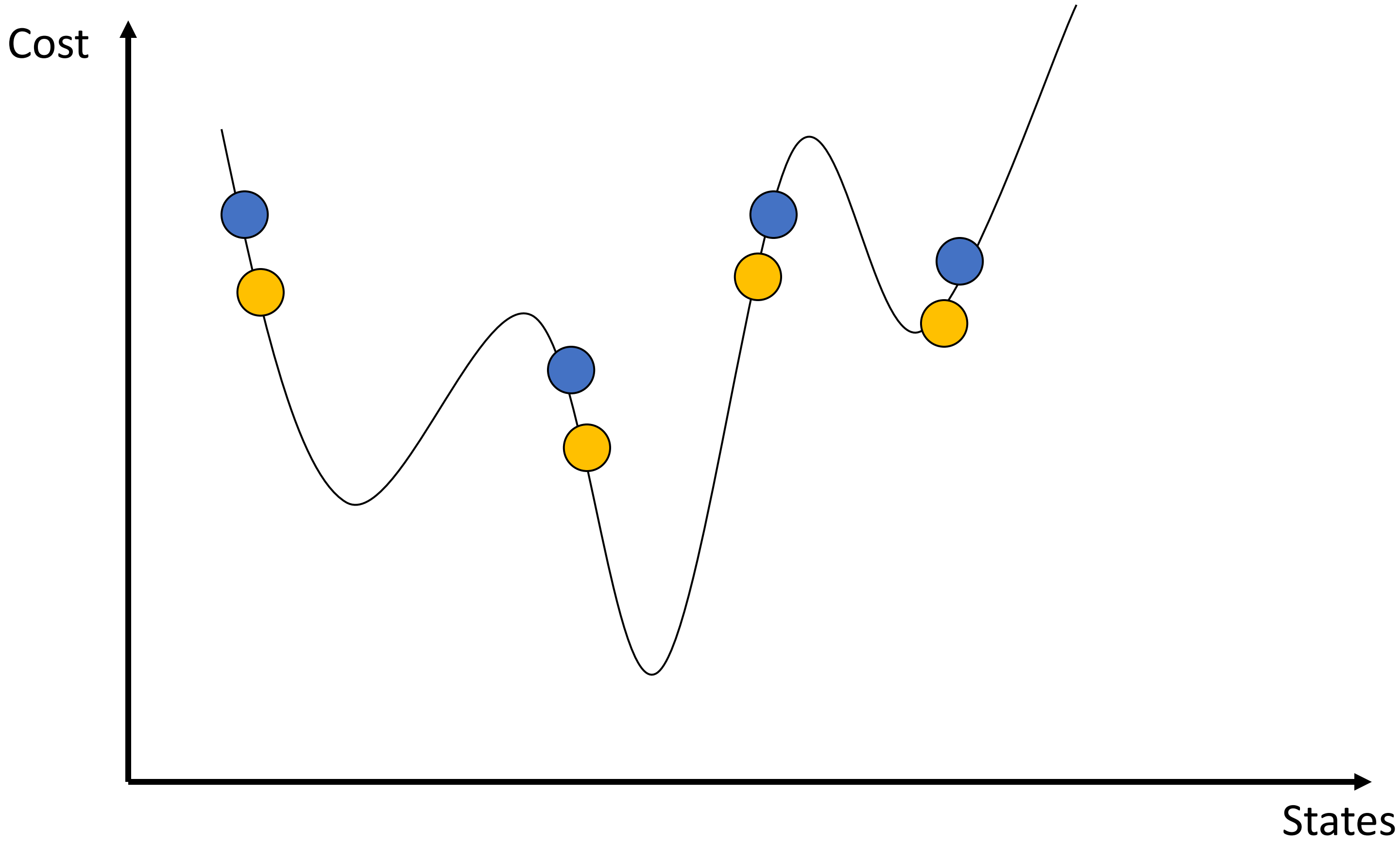
Idea: Keep track of k states rather than just one.

- Start with k randomly generated states
- At each iteration, all the successors of all k states are generated.
  - `If` any one is a goal state, stop;
  - `else` select the k best successors from the complete list and repeat.

3. Local Beam Search

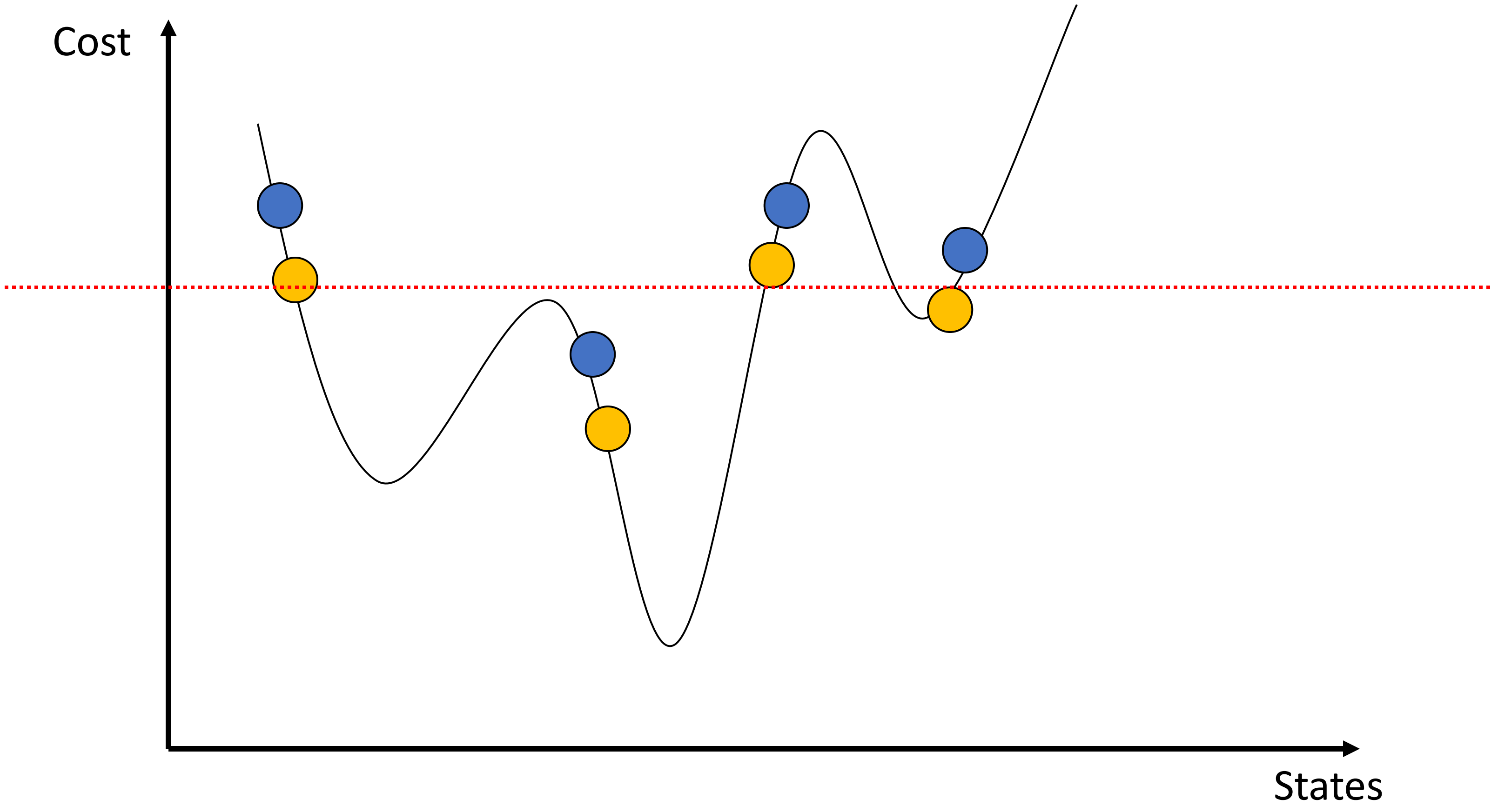


3. Local Beam Search

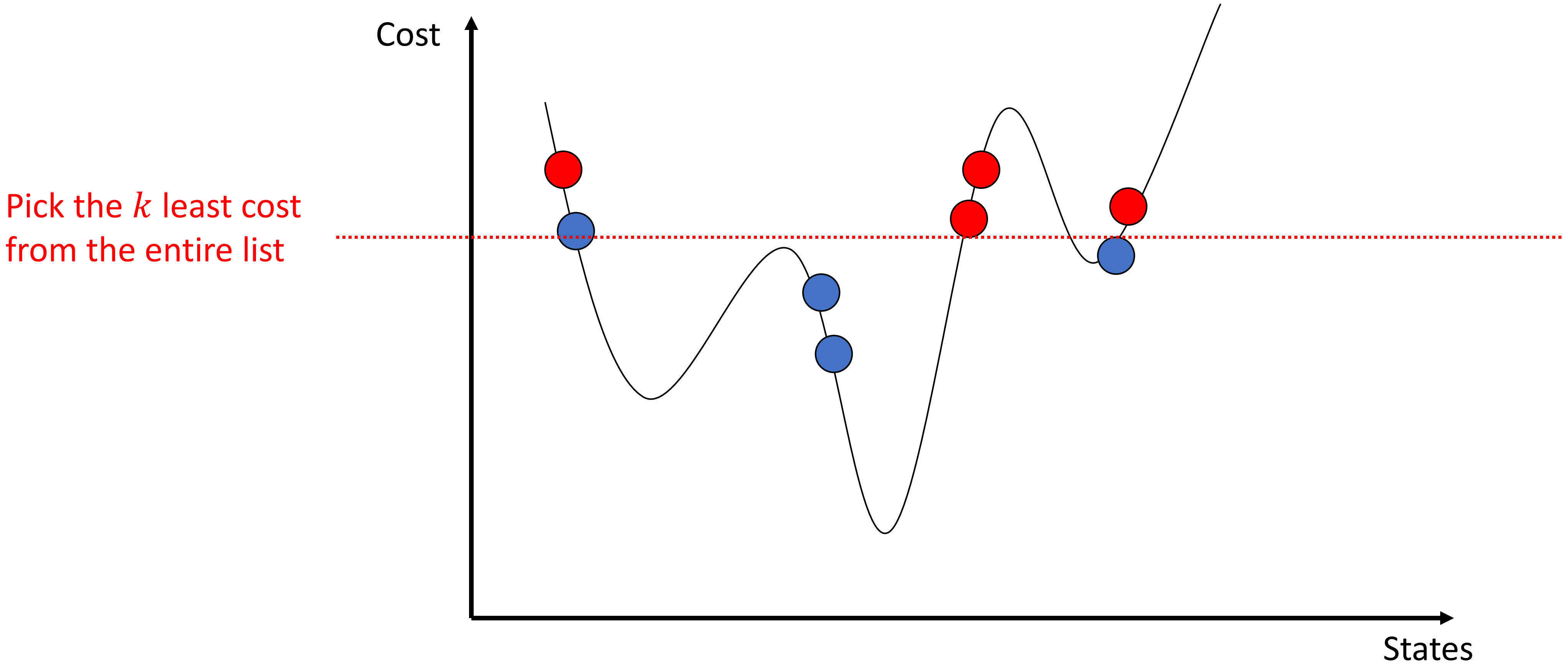


3. Local Beam Search

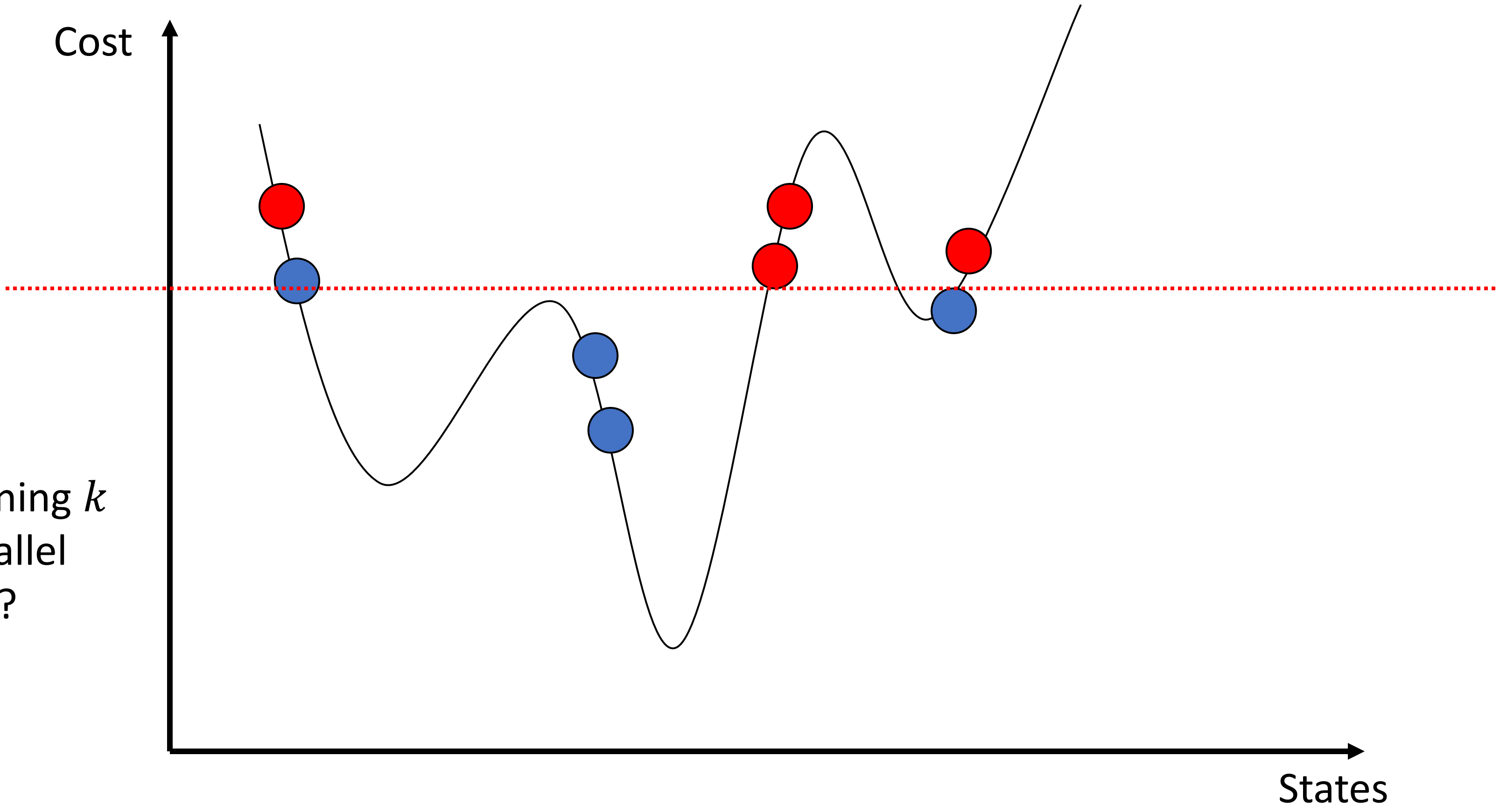
Pick the k least cost from the entire list



3. Local Beam Search



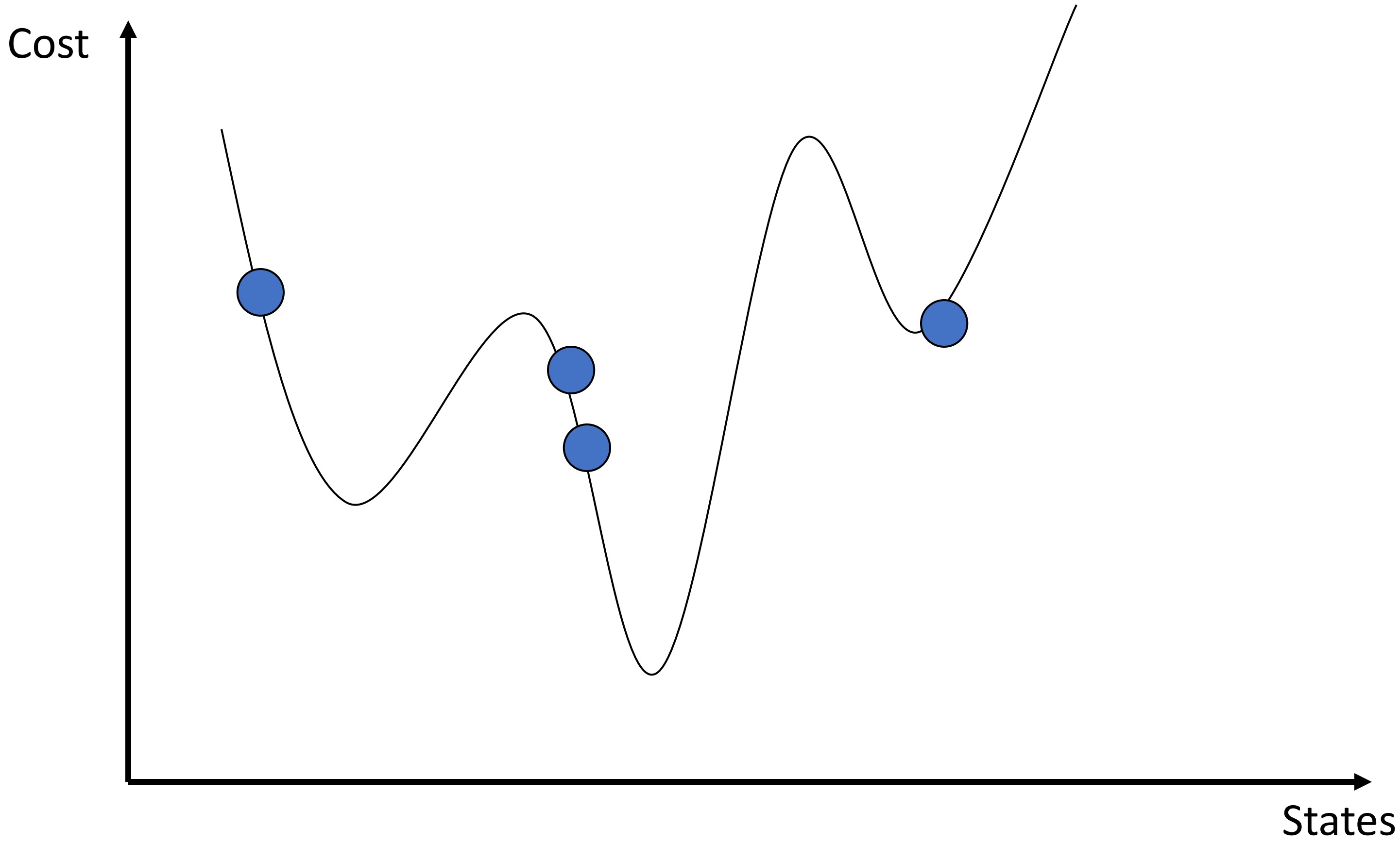
3. Local Beam Search



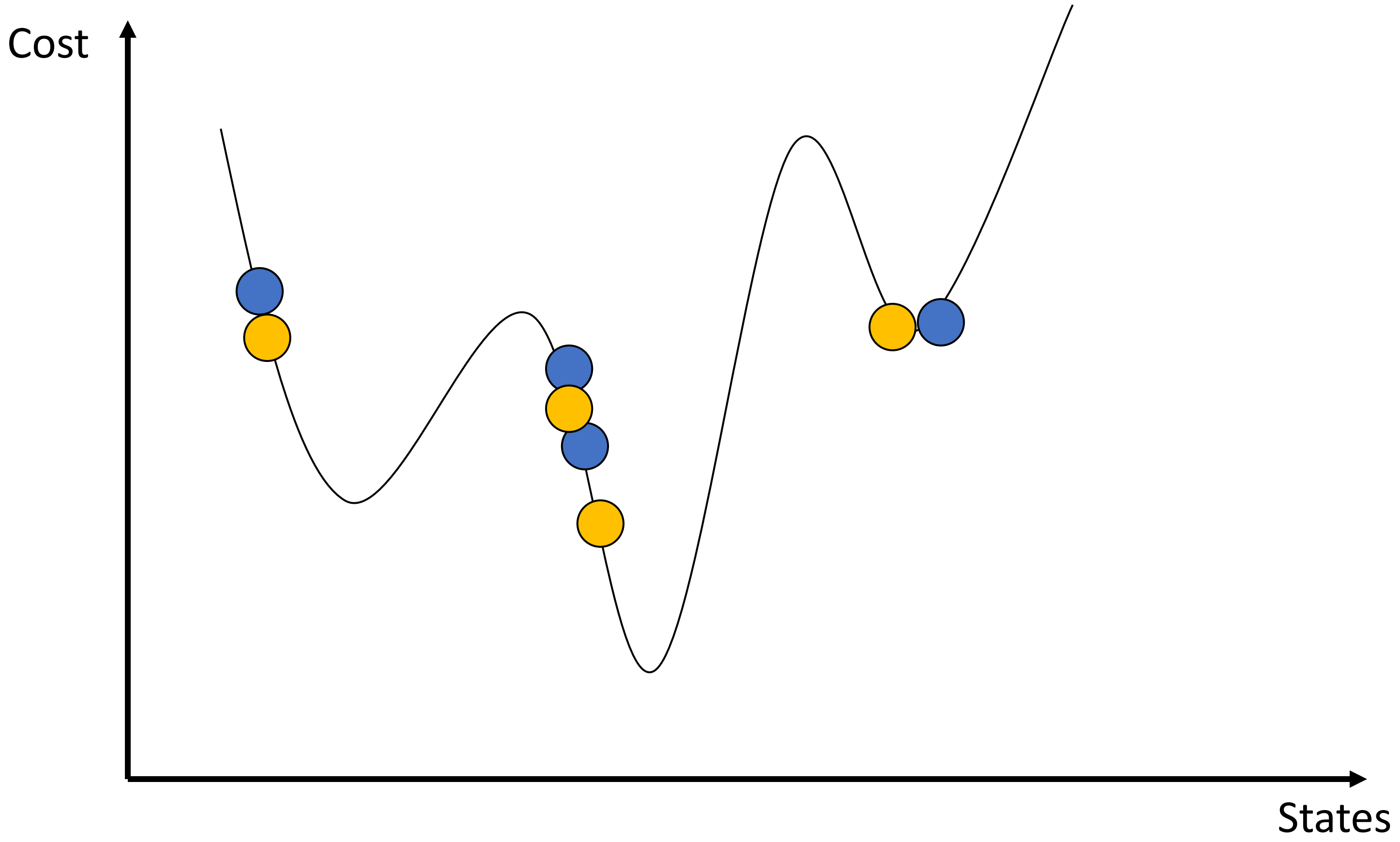
Pick the k least cost from the entire list

Is this the same as running k random restarts in parallel instead of in sequence?

3. Local Beam Search

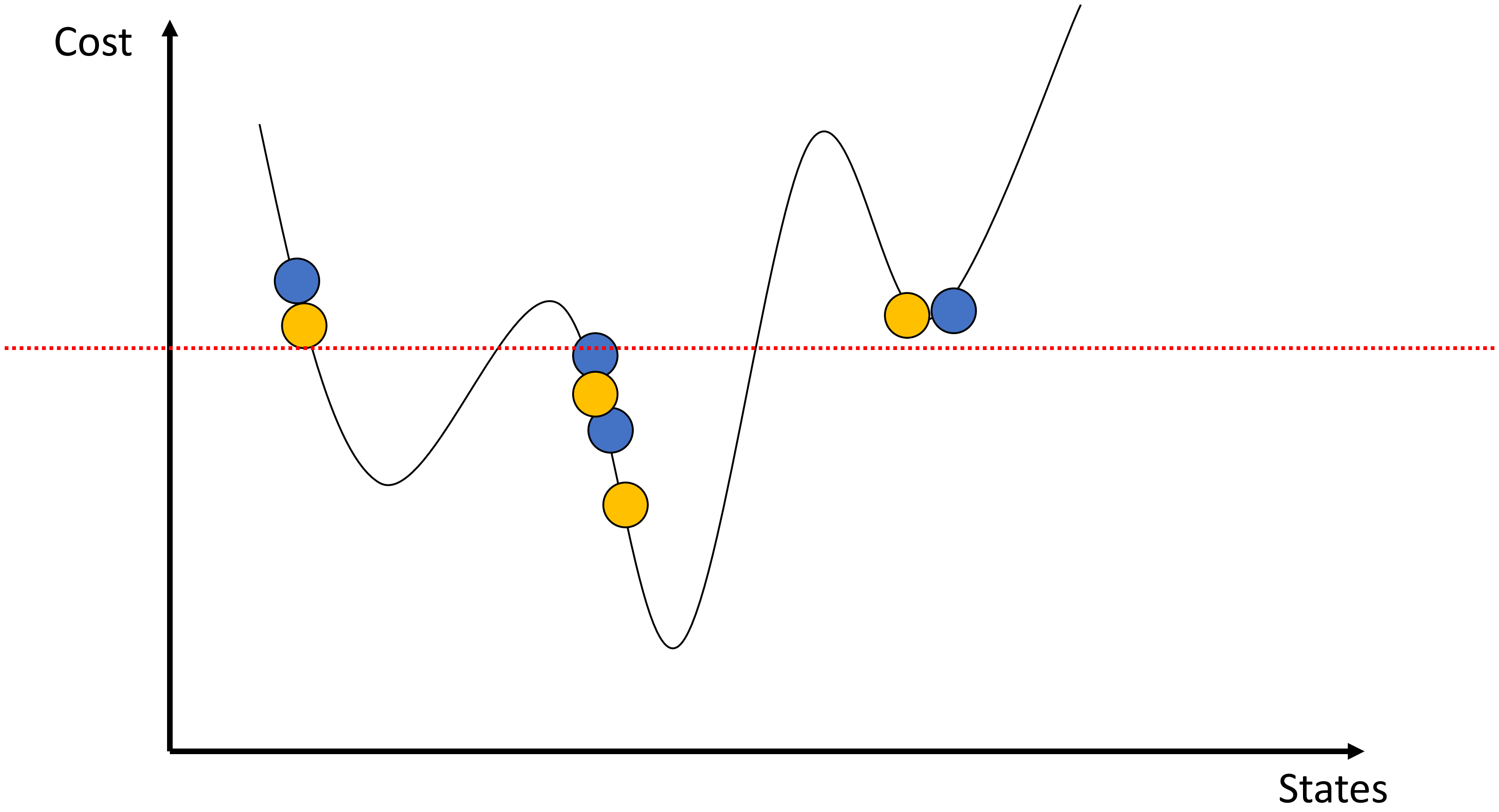


3. Local Beam Search



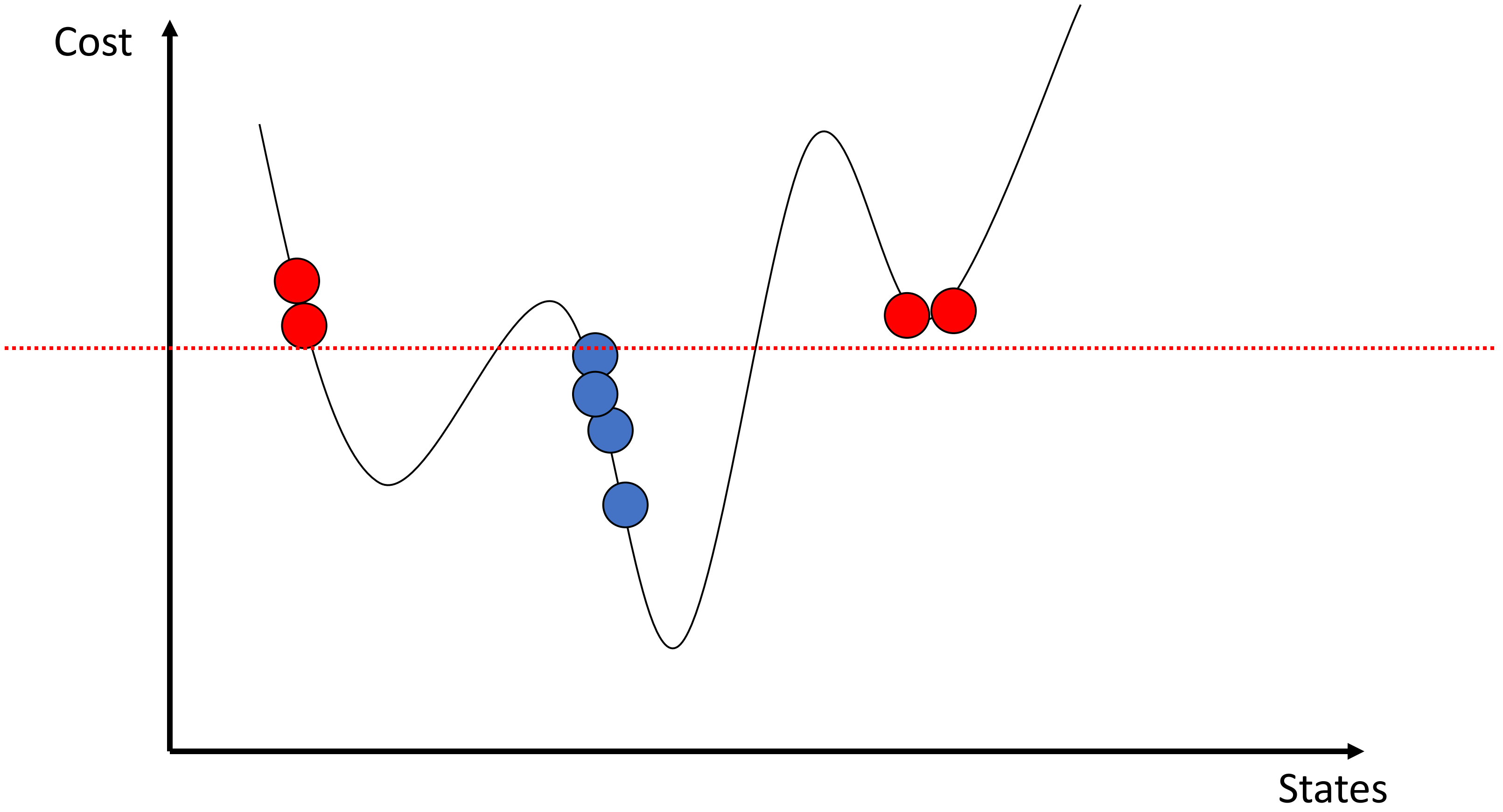
3. Local Beam Search

Pick the k least cost
from the entire list

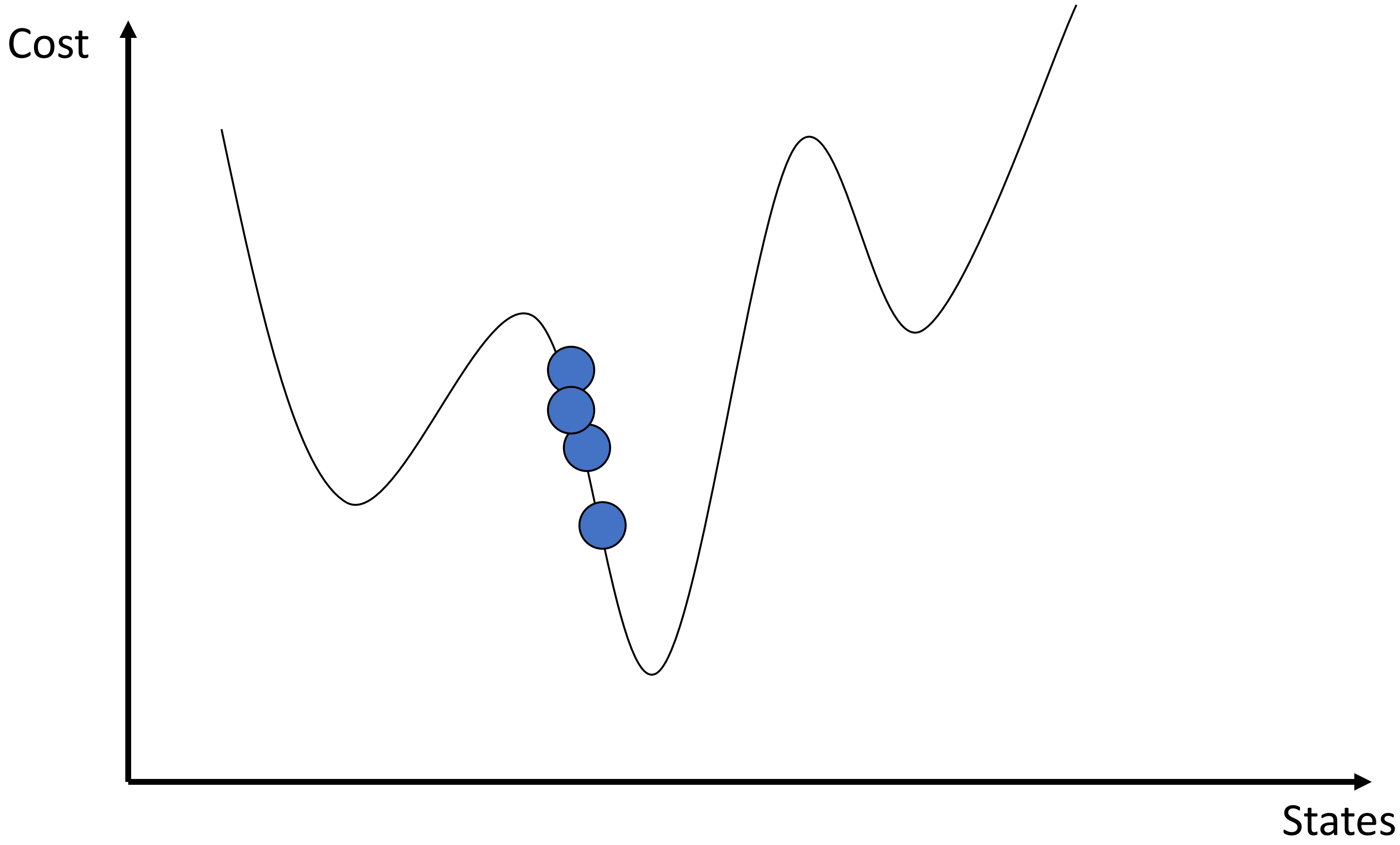


3. Local Beam Search

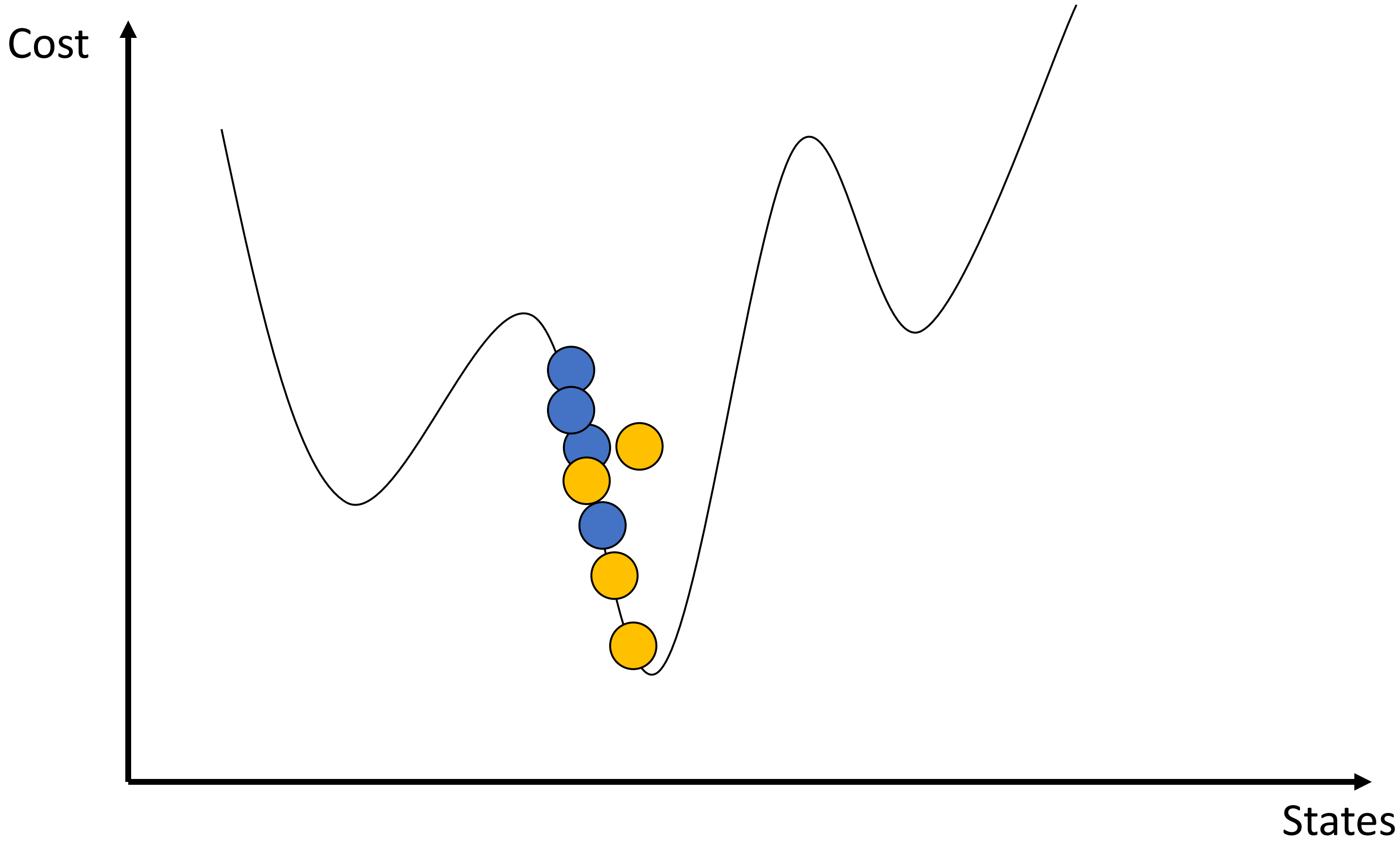
Pick the k least cost from the entire list



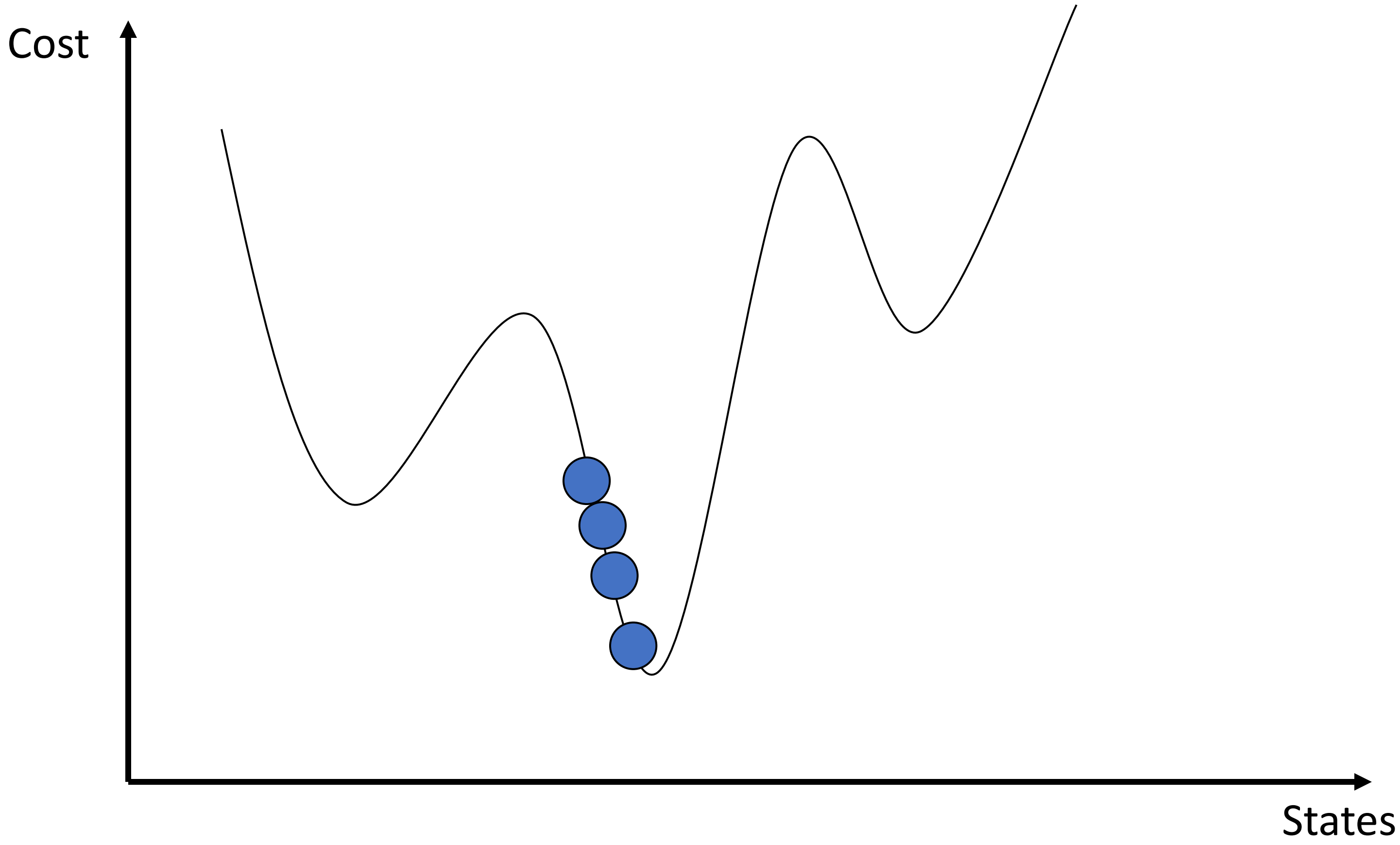
3. Local Beam Search



3. Local Beam Search



3. Local Beam Search



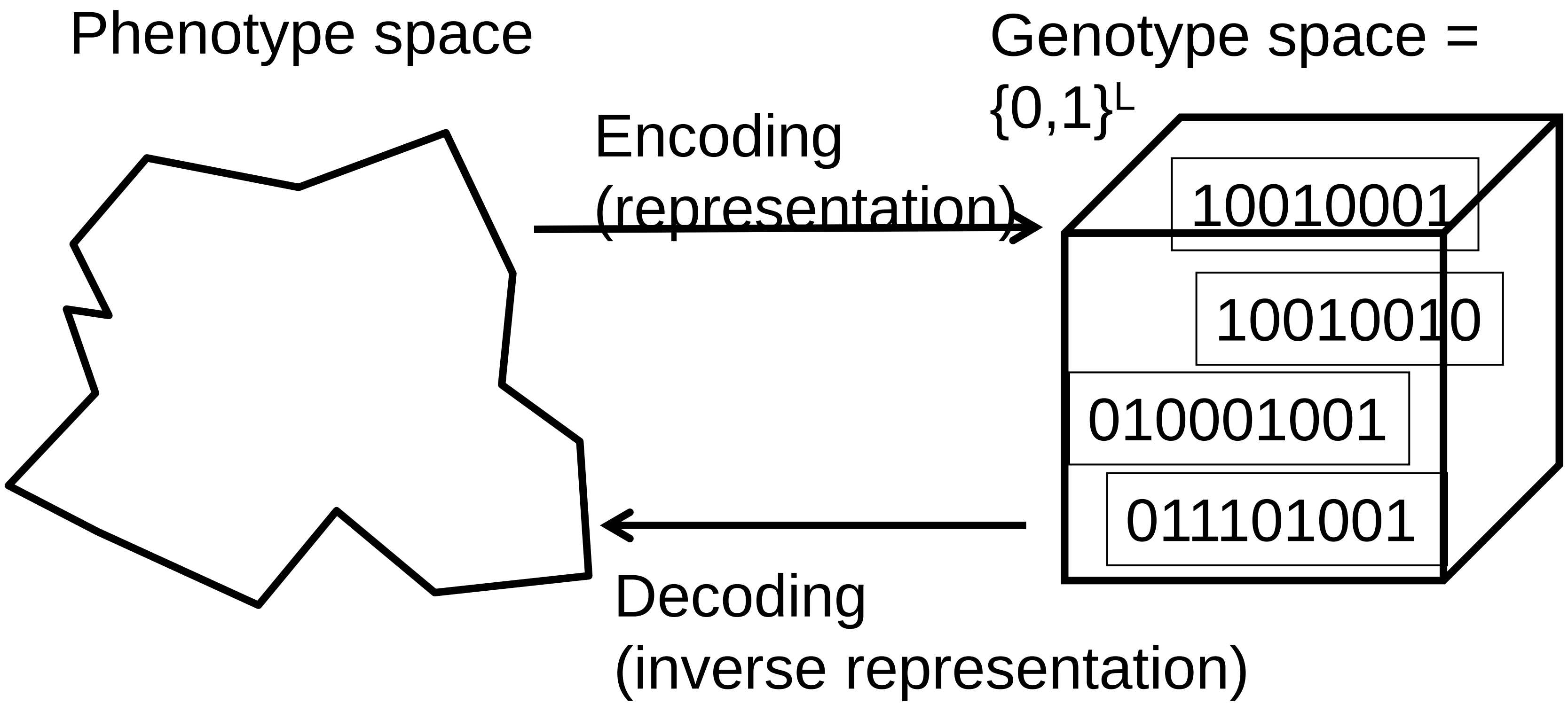
3. Local beam search

- More efficient than hill-climbing.
- However, the k states tend to regroup very quickly in the same region → lack of diversity.
- Improvement: use stochastic methods to generate new state from the old ones.
- **Stochastic beam search:** Instead of choosing the best k from the successors, choose k successors at random, with the probability of choosing a given successor being an increasing function of its value
- Another Example: genetic algorithm.

4. Genetic algorithms

- A successor state is generated by combining **two** parent states
- Start with k randomly generated states (population)
- A **state** (individual) is represented as a string over a finite alphabet (often a string of 0s and 1s)
- **Evaluation function** (fitness function): Higher values for better states.
- Produce the next generation of states by **selection**, **crossover**, and **mutation**.

Representation

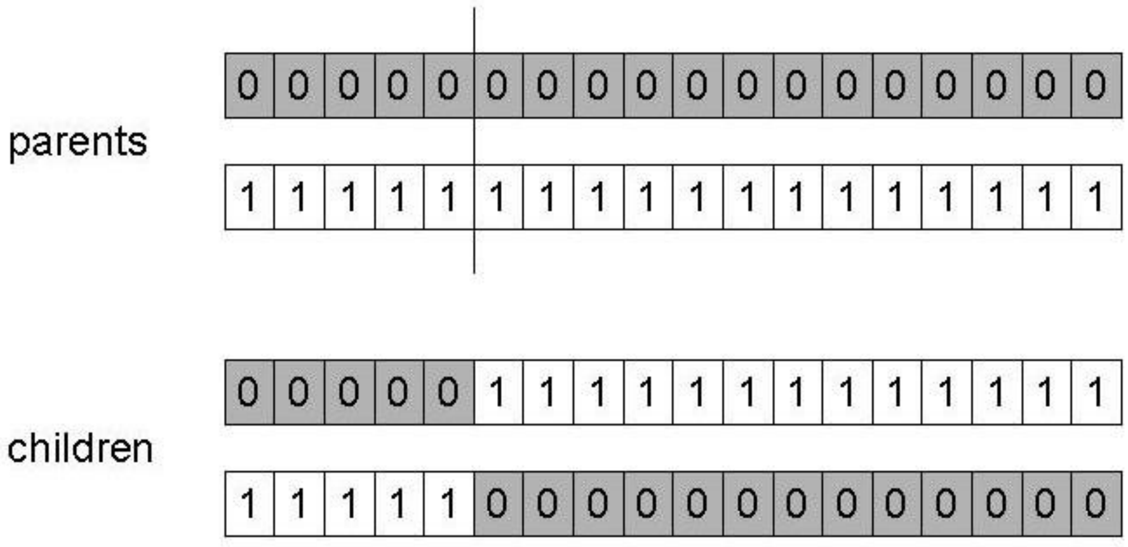


Simple GA reproduction cycle

1. Select parents for the mating pool
   (size of mating pool = population size)
2. Shuffle the mating pool
3. For each consecutive pair apply crossover with probability p_c , otherwise copy parents
4. For each offspring apply mutation (bit-flip with probability p_m independently for each bit)
5. Replace the whole population with the resulting offspring

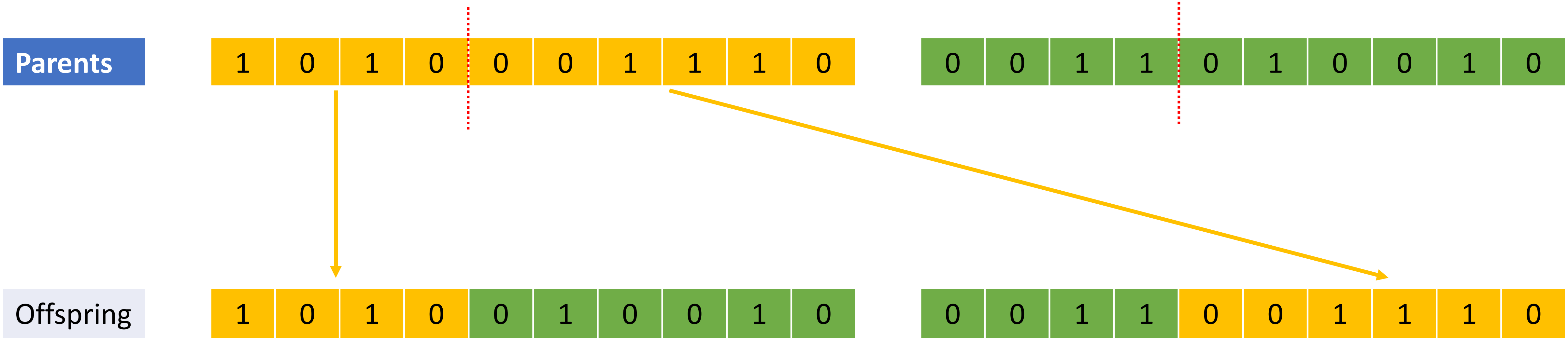
SGA operators: 1-point crossover

- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails
- P_c typically in range (0.6, 0.9)



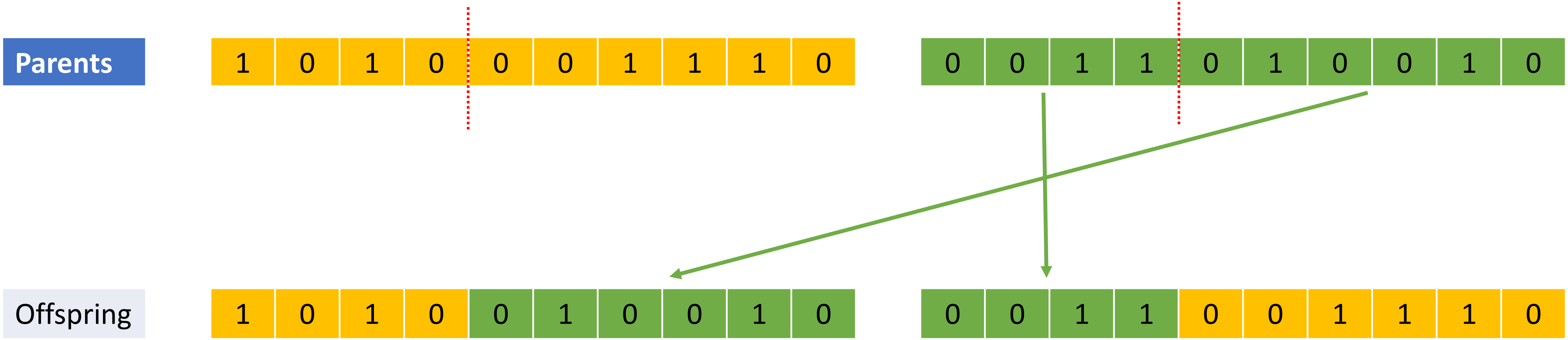
One-point crossover

- Randomly choose one position in the chromosomes



One-point crossover

- Randomly choose one position in the chromosomes



Two-point crossover

- Randomly choose two positions in the chromosomes

Parents	1	0	1	0	0	0	1	1	1	0	0	0	1	1	0	1	0	0	1	0
Offspring	1	0	1	0	0	1	0	1	1	0	0	0	1	1	0	0	1	0	1	0

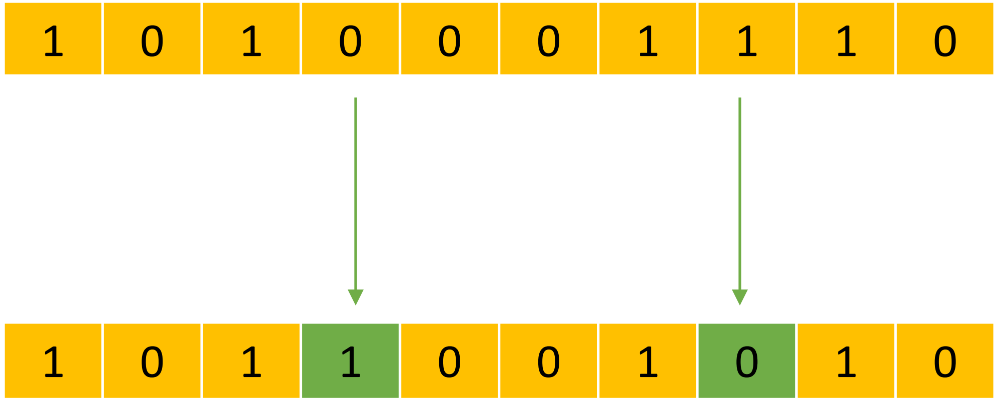
SGA operators: mutation

- Alter each gene independently with a probability p_m
- p_m is called the mutation rate
    - Typically between 1/pop_size and 1/ chromosome_length

parent	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
child	0	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	0	1

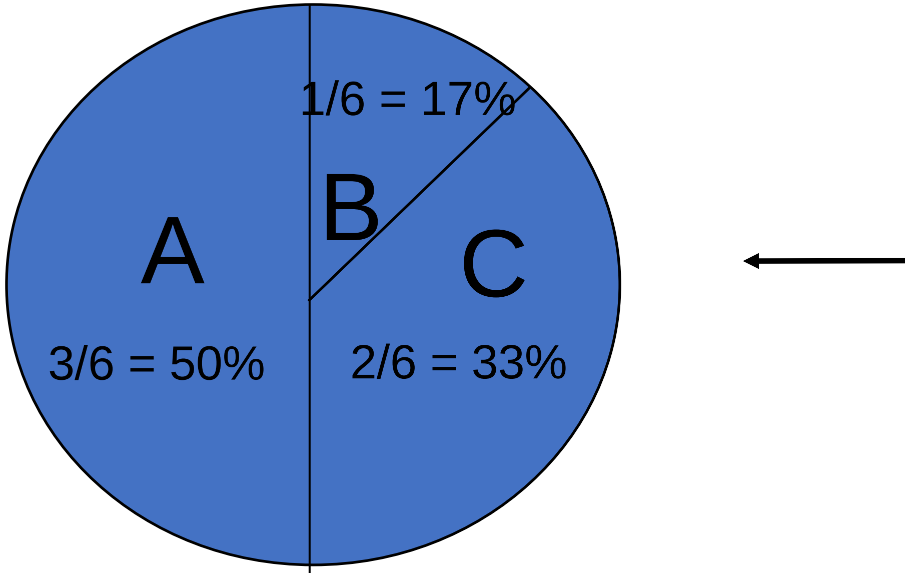
Mutation

- There are different ways to perform mutation
- The idea is to introduce a small change
  - Replace a bit by its complement



SGA operators: Selection

- Main idea: better individuals get higher chance
  - Chances proportional to fitness
  - Implementation: roulette wheel technique
    - Assign to each individual a part of the roulette wheel
    - Spin the wheel n times to select n individuals



fitness(A) = 3

fitness(B) = 1

fitness(C) = 2

Example

- Simple problem: max x^2 over $\{0,1,\ldots,31\}$
- GA approach:
  - Representation: binary code, e.g. 01101 $\leftrightarrow$ 13
  - Population size: 4
  - 1-point xover, bitwise mutation
  - Roulette wheel selection
  - Random initialisation
- We show one generational cycle done by hand

x^2 example: selection

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

X^2 example: crossover

String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0	4	1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

X^2 example: mutation

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	$\boxed{1}$ 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 $\boxed{1}$ 0 0	18	324
Sum				2354
Average				588.5
Max				729

The simple GA

- Has been subject of many (early) studies
  - still often used as benchmark for novel GAs
- Shows many shortcomings, e.g.
  - Representation is too restrictive
  - Mutation & crossovers only applicable for bit-string & integer representations
  - Selection mechanism sensitive for converging populations with close fitness values
  - Generational population model (step 5 in SGA repr. cycle) can be improved with explicit survivor selection

4. Representation: Traveling Salesman Problem

- The Find a tour of a given set of cities so that each city is visited only once, and the total distance traveled is minimized
- Representation is an ordered list of city numbers.

1) London	3) Dunedin	5) Beijing	7) Tokyo
2) Venice	4) Singapore	6) Phoenix	8) Victoria

Chromosome1 (3 5 7 2 1 6 4 8)

Chromosome2 (2 5 7 6 8 1 3 4)

4. Representation: 8-queens

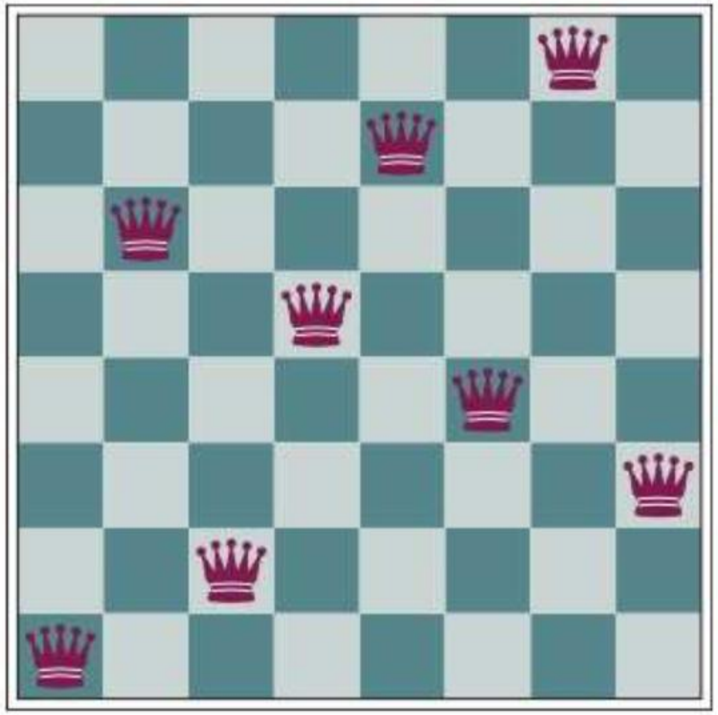
- An 8-queens state must specify the positions of 8 queens, each in a column of 8 squares, and so requires $8 \times \log_2 8 = 24$ bits

  8 queens   3 bits to specify 8 positions

- Alternatively, the state could be represented as 8 digits, each in the range from 1 to 8.
- The two encodings behave differently.
- Successful use of GA requires careful engineering of the representation

4. Representation: 8-queens

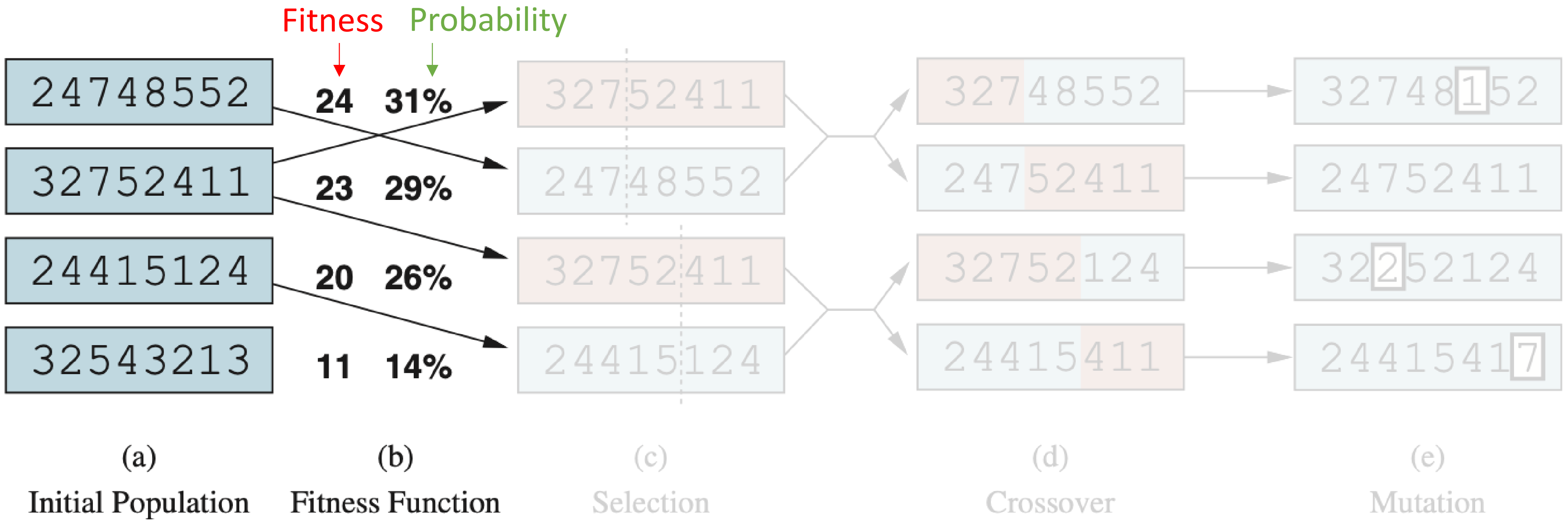
Individual = 1 6 2 5 7 4 8 3



4. Fitness Function: 8-queens

- A fitness function should return higher values for better states
- 8-queens problem: we use the number of *nonattacking* pairs of queens
  - has a value of 28 for a solution (7+6+5+4+3+2+1)
- Calculate the fitness of every individual in the population
- The probability of being chosen for reproducing is directly proportional to the fitness score

4. Fitness Function: 8-queens



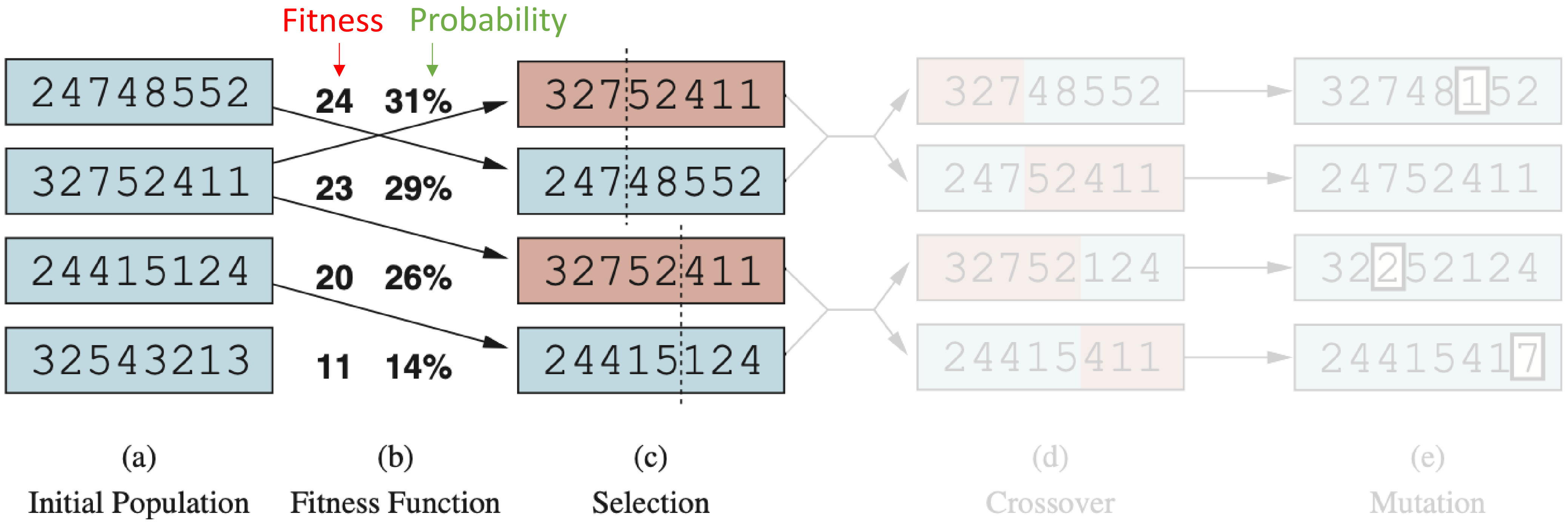
$$\frac{24}{11+20+23+24} = \frac{24}{78} = 0.308$$

$$\frac{20}{11+20+23+24} = \frac{20}{78} = 0.256$$

$$\frac{23}{11+20+23+24} = \frac{23}{78} = 0.295$$

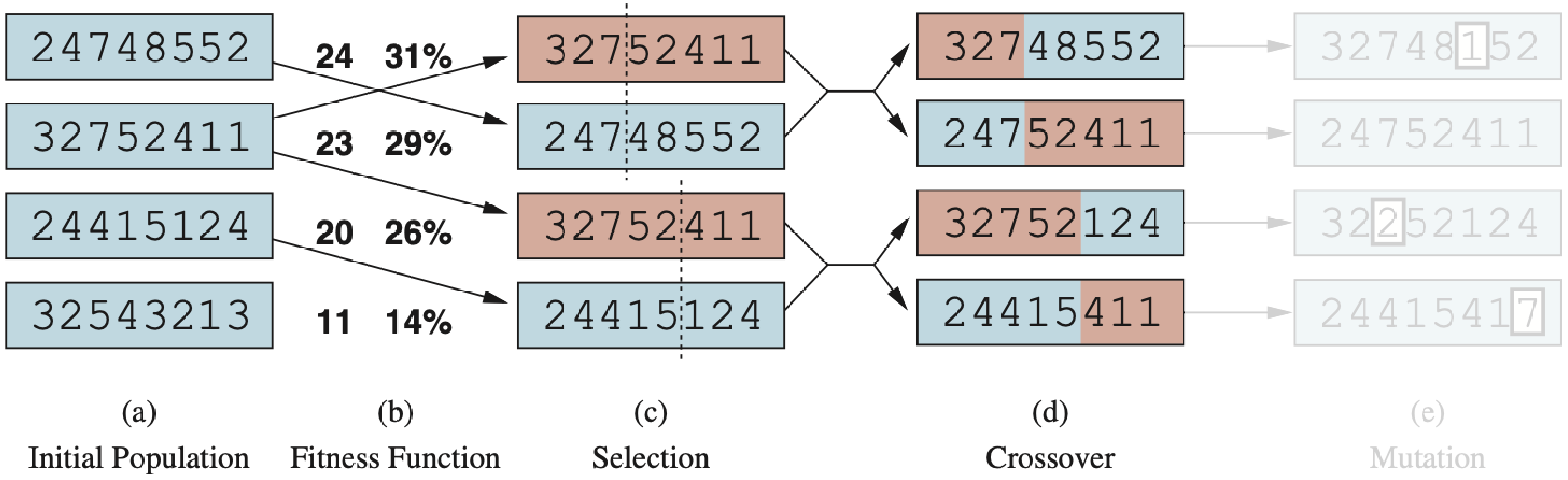
$$\frac{11}{11+20+23+24} = \frac{11}{78} = 0.141$$

4. Selection: 8-queens



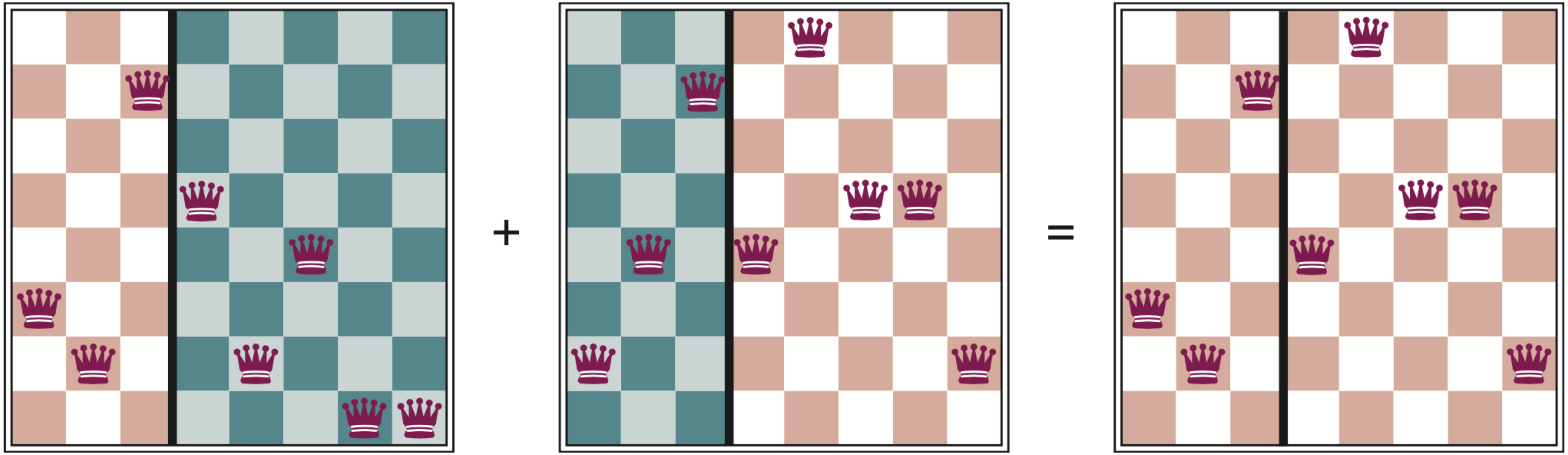
- Two pairs are selected at random for reproduction, in accordance with the probabilities in (b)
- Notice that one individual is selected twice and one not at all

4. Crossover: 8-queens

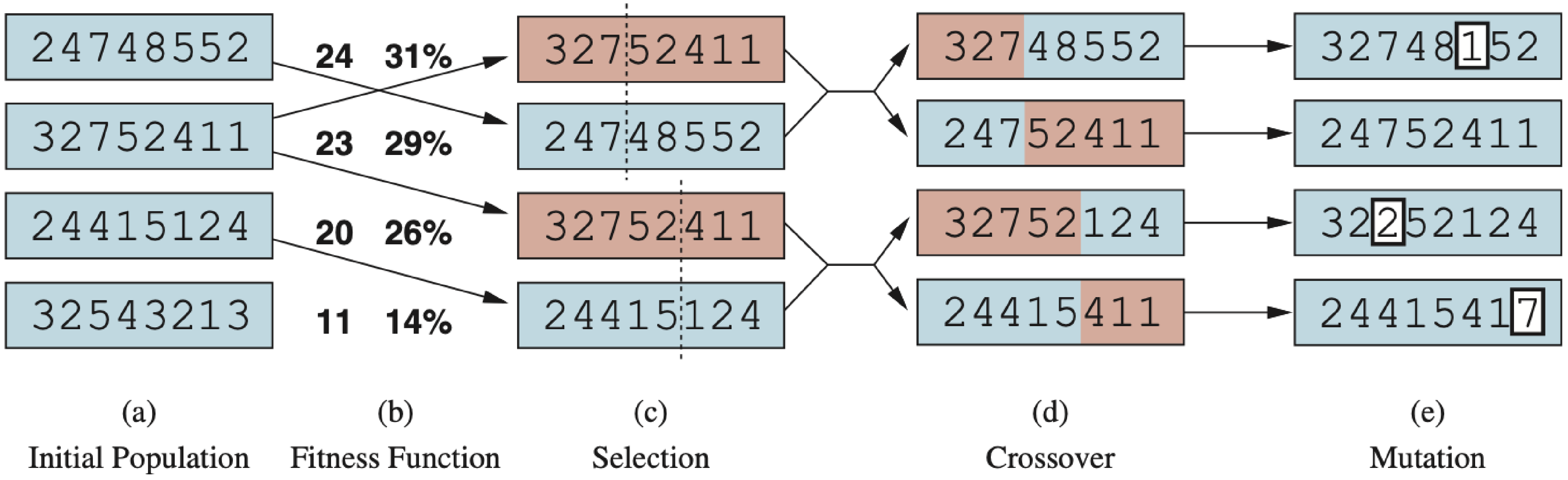


- A **crossover** point is chosen randomly from the positions in the string

4. Crossover: 8-queens



4. Mutation: 8-queens



- Each location is subject to random **mutation** with a small independent probability
- Mutation probability is low (e.g., 0.001)

GA: how does it work?

- Replace the old population with the new population
- Repeat the previous steps until the best individual is found or the maximum number of iterations is reached

GA algorithm

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
      weights ← WEIGHTED-BY(population, fitness)
      population2 ← empty list
      for i = 1 to SIZE(population) do
          parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
          child ← REPRODUCE(parent1, parent2)
          if (small random probability) then child ← MUTATE(child)
          add child to population2
      population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

Genetic algorithms

- The **goal** of crossover is local search: look for new individuals that are similar to the best individuals in the population.
  - This is called exploitation
- The **goal** of mutation is to explore new parts of the search space.
  - This is called exploration