Local Search

Instructor

LE Thanh Sach, Ph.D.

Instructor's Information

LE Thanh Sach, Ph.D.

Office:

Department of Computer Science,
Faculty of Computer Science and Engineering,
HoChiMinh City University of Technology.

Office Address:

268 LyThuongKiet Str., Dist. 10, HoChiMinh City, Vietnam.

E-mail: LTSACH@hcmut.edu.vn

E-home: http://cse.hcmut.edu.vn/~ltsach/

Tel: (+84) 83-864-7256 (Ext: 5839)

Acknowledgment

The slides in this PPT file are composed using the materials supplied by

- **Prof. Stuart Russell and Peter Norvig**: They are currently from University of California, Berkeley. They are also the author of the book "Artificial Intelligence: A Modern Approach", which is used as the textbook for the course

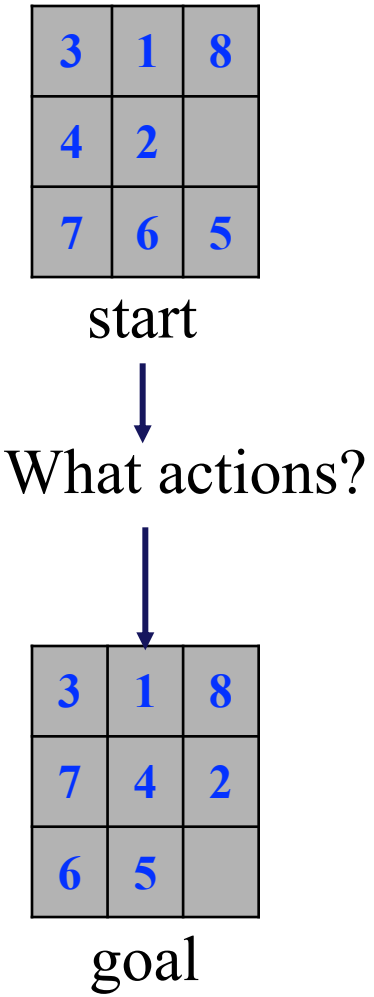
- **Prof. Tom Lenaerts**, from Université Libre de Bruxelles

Outline

- Recap
- Local Search - Introduction
- Hill Climbing
- Simulated Annealing
- Genetic Algorithms

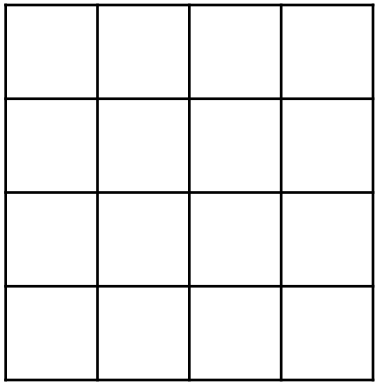
Recap

- Previously strategies
  - Breadth-First Search
  - Depth-First Search
  - Uniform-Cost Search
  - Greedy Best-First Search
  - A-Star
  - …
- Previously strategies' type of solutions
  - A typical solution is a **path** from the start to the goal state



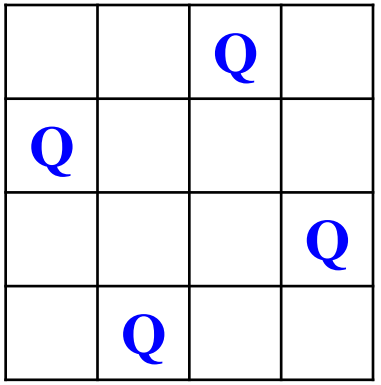
Recap

- Previously strategies
  - Breadth-First Search
  - Depth-First Search
  - Uniform-Cost Search
  - Greedy Best-First Search
  - A-Star
  - …
- Previously strategies' type of solutions
  - A typical solution is a **path** from the start to the goal state



start

What actions?

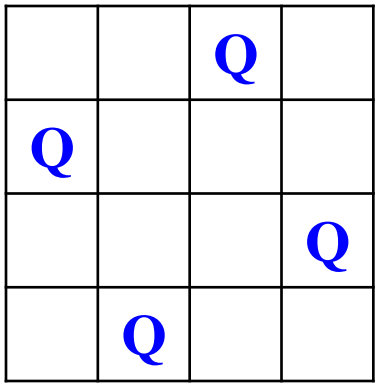


a goal

Local search and optimization

Local Search

- Local search's type of solutions
  - NOT a path from the start to the goal state
  - **Solution: a state, a representation that satisfies users' need**
  - **The start state???**
    - ✓ Any (a randomly generated state)

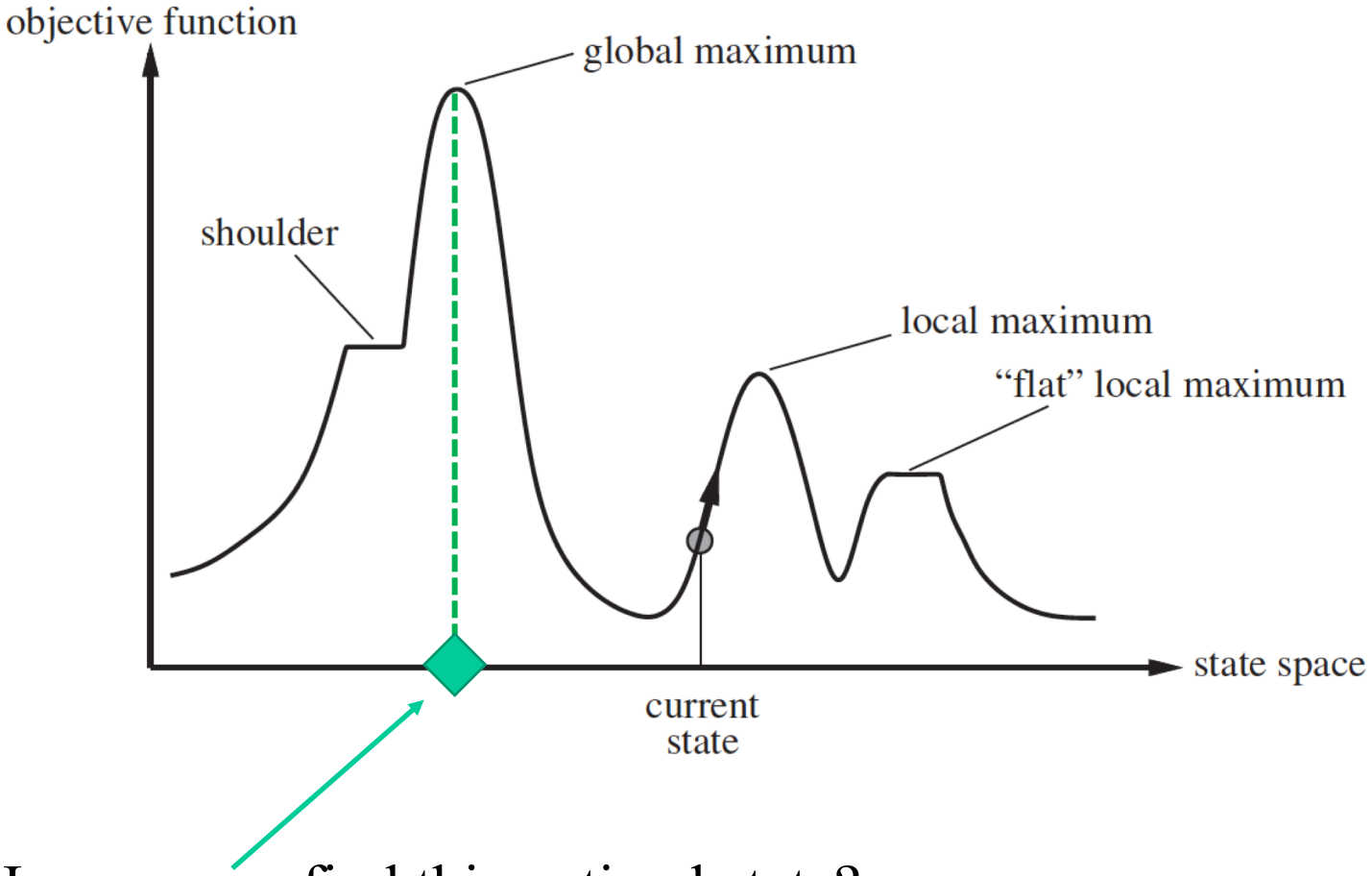


a goal

Local search and optimization

- Local search= use single current state and move to neighboring states.
- Advantages:
  - Use very little memory
  - Find often reasonable solutions in large or infinite state spaces.
- Are also useful for pure optimization problems.
  - Find best state according to some *objective function*.
  - e.g. survival of the fittest as a metaphor for optimization.

Local search and optimization



How can we find this optimal state?

Hill-climbing

Hill-climbing search

- "is a loop that continuously moves in the direction of increasing value"
  - It terminates when a peak is reached.
- Hill climbing does not look ahead of the immediate neighbors of the current state.
- Hill-climbing chooses randomly among the set of best successors, if there is more than one.
- Hill-climbing a.k.a. *greedy local search*

Hill-climbing search

function HILL-CLIMBING(*problem*) **return** a state that is a local maximum

input: *problem*, a problem

local variables: *current*, **a node.**

neighbor, **a node.**

current ← MAKE-NODE(INITIAL-STATE[*problem*])

loop do

neighbor ← a highest valued successor of *current*

if VALUE [*neighbor*] ≤ VALUE[*current*] **then return** STATE[*current*]

current ← *neighbor*

Hill-climbing search

function HILL-CLIMBING(*problem*) **return** a state that is a local maximum

input: *problem*, a problem

local variables: *current*, **a node.**

neighbor, **a node.**

current ← MAKE-NODE(INITIAL-STATE[*problem*])

loop do

neighbor ← a highest valued successor of *current*

if VALUE [*neighbor*] ≤ VALUE[*current*] **then return** STATE[*current*]

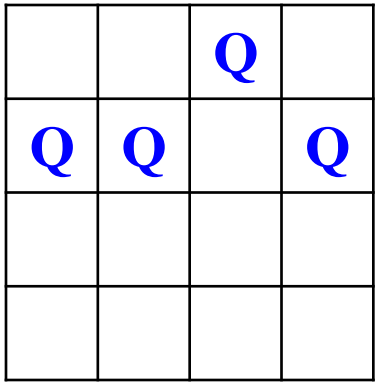
current ← *neighbor*

We are at a peak already → stop searching

Hill-climbing example

- 8-queens problem (complete-state formulation).
- Successor function: move a single queen to another square in the same column.
- Heuristic function *h(n)*: the number of pairs of queens that are attacking each other (directly or indirectly).

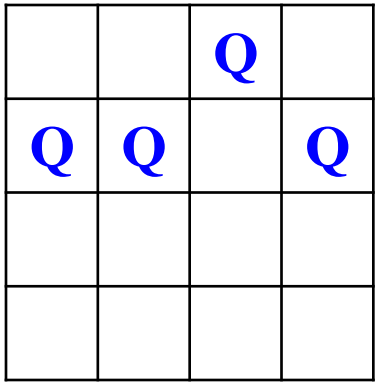
Hill-climbing example



A start state

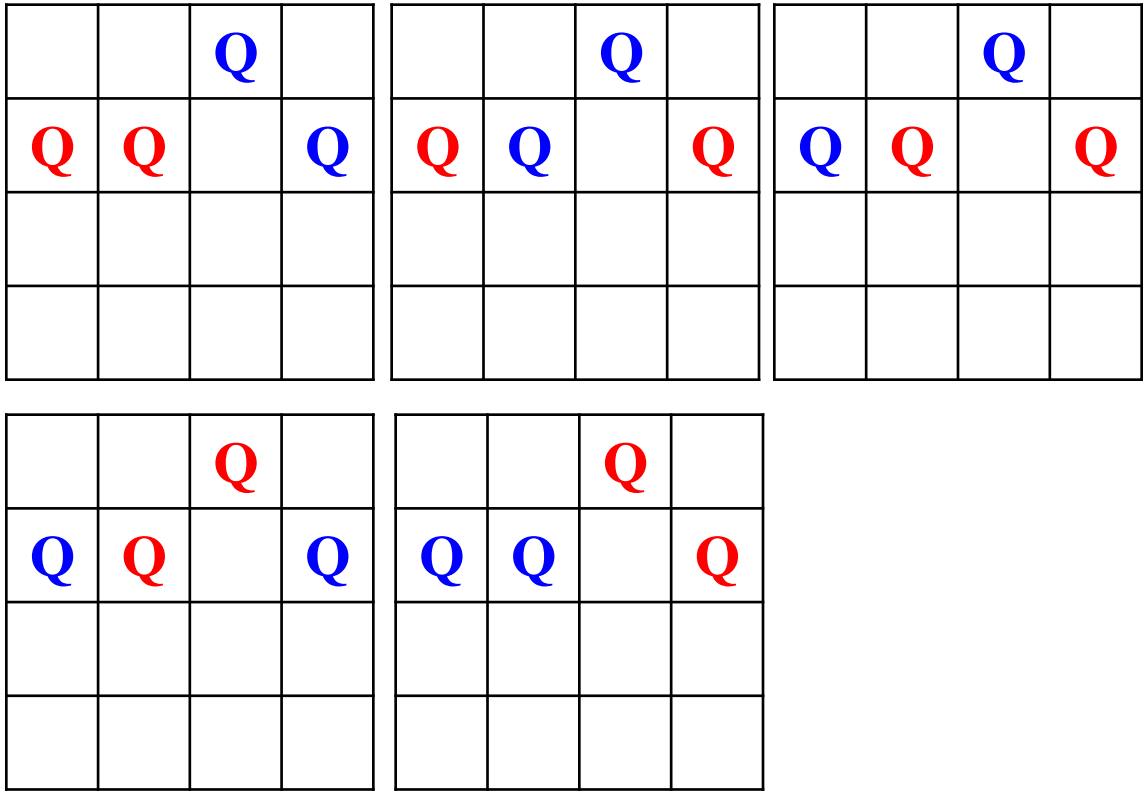
(Assume that we are given this state to start)

Hill-climbing, objective function

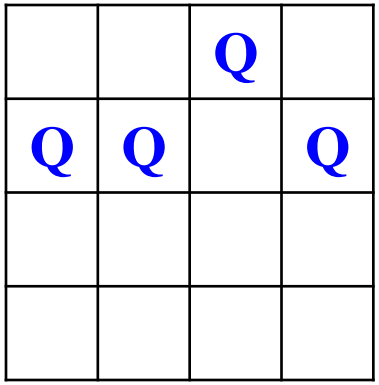


A start state

#attackingpairs: 5



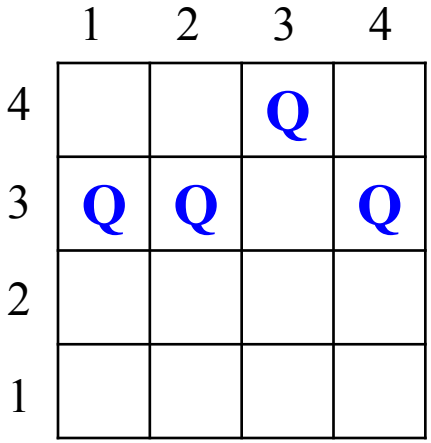
Hill-climbing, objective function



A start state

How many neighbours does this have?

Hill-climbing, objective function



A start state

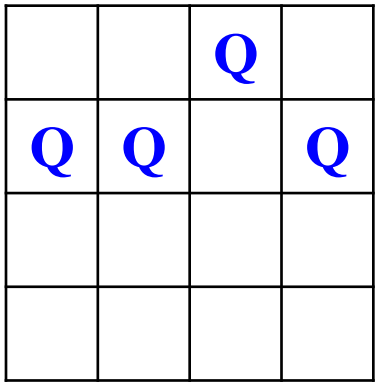
How many neighbours does this have?

There are four queens; each moving in a column

Three possible moves for each queens

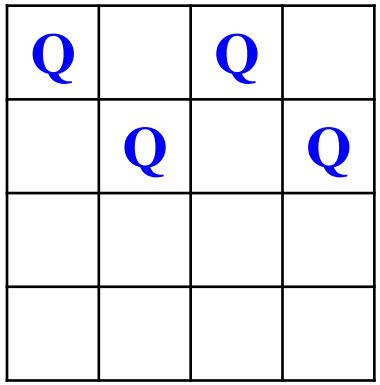
➔ 4x3 = 12 neighbours (corresponding to 12 blank cells)

Hill-climbing, objective function



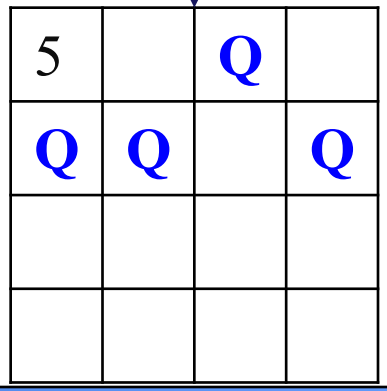
A start state
#attackingpairs: 5

Move
the first queen to (4,1)

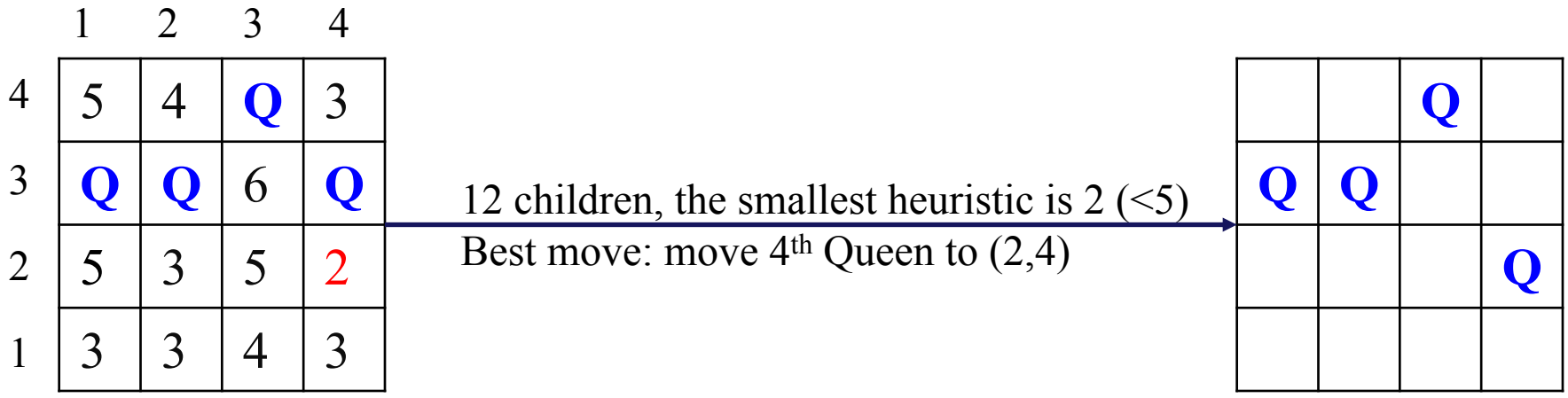


A start state
#attackingpairs: 5

Represented



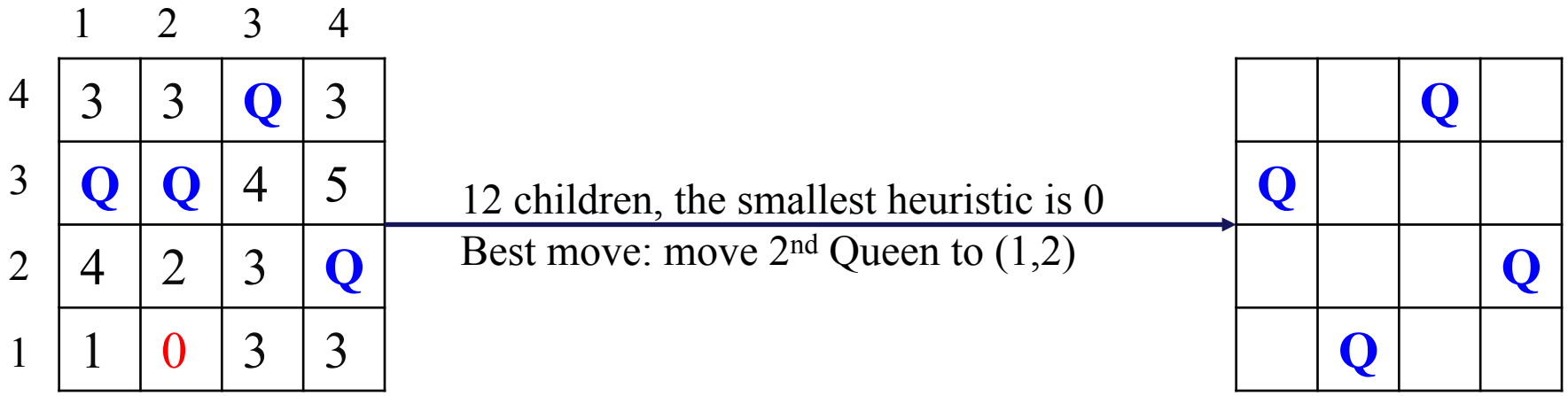
Hill-climbing, step-by-step



A start state
#attackingpairs: 5

#attackingpairs: 2

Hill-climbing, step-by-step



#attackingpairs: 2

#attackingpairs: 0

Solution is found

The solution can be found after 2 moves

Hill-climbing, step-by-step

3	3	**Q**	3
Q	**Q**	4	5
4	2	3	**Q**
1	0	3	3

#attackingpairs: 2

12 children, the smallest heuristic is 0

➔ Make a move to the solution

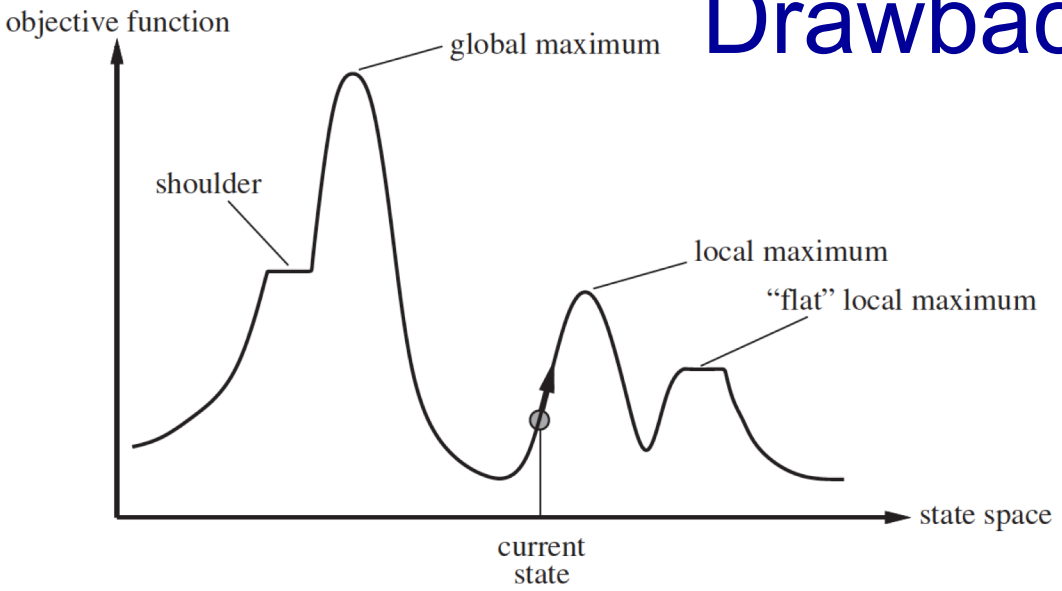
		Q	
Q			
			Q
	Q		

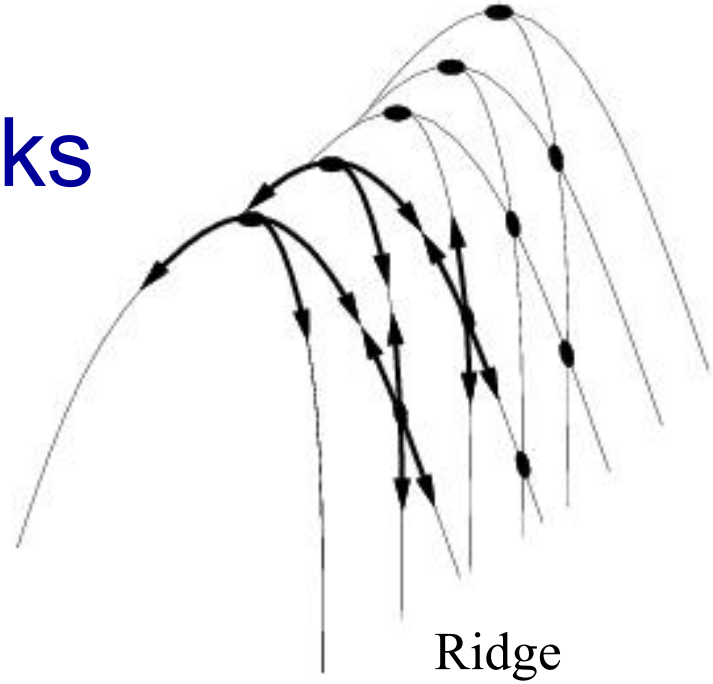
#attackingpairs: 0

Solution is found

The solution can be found after 2 moves
Lucky, most of the cases we can not solve the game
by using simple Hill-Climbing

Drawbacks





- Ridge = sequence of local maxima difficult for greedy algorithms to navigate
- Plateaux = an area of the state space where the evaluation function is flat.
- Gets stuck 86% of the time.

Hill-climbing variations

- Stochastic hill-climbing
  - Random selection among the uphill moves.
  - The selection probability can vary with the steepness of the uphill move.
- First-choice hill-climbing
  - cfr. stochastic hill climbing by generating successors randomly until a better one is found.
- Random-restart hill-climbing
  - Tries to avoid getting stuck in local maxima.

Simulated annealing

Simulated annealing

- Escape local maxima by allowing "bad" moves.
  - Idea: but gradually decrease their size and frequency.
- Origin; metallurgical annealing
- Bouncing ball analogy:
  - Shaking hard (= high temperature).
  - Shaking less (= lower the temperature).
- If T decreases slowly enough, best state is reached.
- Applied for VLSI layout, airline scheduling, etc.

Simulated annealing

function SIMULATED-ANNEALING(*problem, schedule*) **return** a solution state

input: *problem*, a problem

schedule, a mapping from time to temperature

local variables: *current*, a node.

next, a node.

T, a "temperature" controlling the probability of downward steps

current ← MAKE-NODE(INITIAL-STATE[*problem*])

for t ← 1 to ∞ do

T ← *schedule*[*t*]

if *T* = *0* **then return** *current*

next ← a randomly selected successor of *current*

Δ*E* ← VALUE[*next*] - VALUE[*current*]

if Δ*E* > 0 **then** *current* ← *next*

else *current* ← *next* only with probability $e^{\Delta E/T}$

Simulated annealing

function SIMULATED-ANNEALING(*problem, schedule*) **return** a solution state

input: *problem*, a problem

schedule, a mapping from time to temperature

local variables: *current*, a node.

next, a node.

T, a "temperature" controlling the probability of downward steps

current ← MAKE-NODE(INITIAL-STATE[*problem*])

for t ← 1 to ∞ do

T ← *schedule*[*t*]

Temperature T: decreased among time t

if *T* = *0* **then return** *current*

next ← a randomly selected successor of *current*

Δ*E* ← VALUE[*next*] - VALUE[*current*]

if Δ*E* > 0 **then** *current* ← *next*

else *current* ← *next* only with probability $e^{\Delta E / T}$

Simulated annealing

function SIMULATED-ANNEALING(*problem, schedule*) **return** a solution state

input: *problem*, a problem

schedule, a mapping from time to temperature

local variables: *current*, a node.

next, a node.

T, a "temperature" controlling the probability of downward steps

current ← MAKE-NODE(INITIAL-STATE[*problem*])

for t ← 1 to ∞ do

T ← *schedule*[*t*]

if *T* = *0* **then return** *current*

next ← a randomly selected successor of *current*

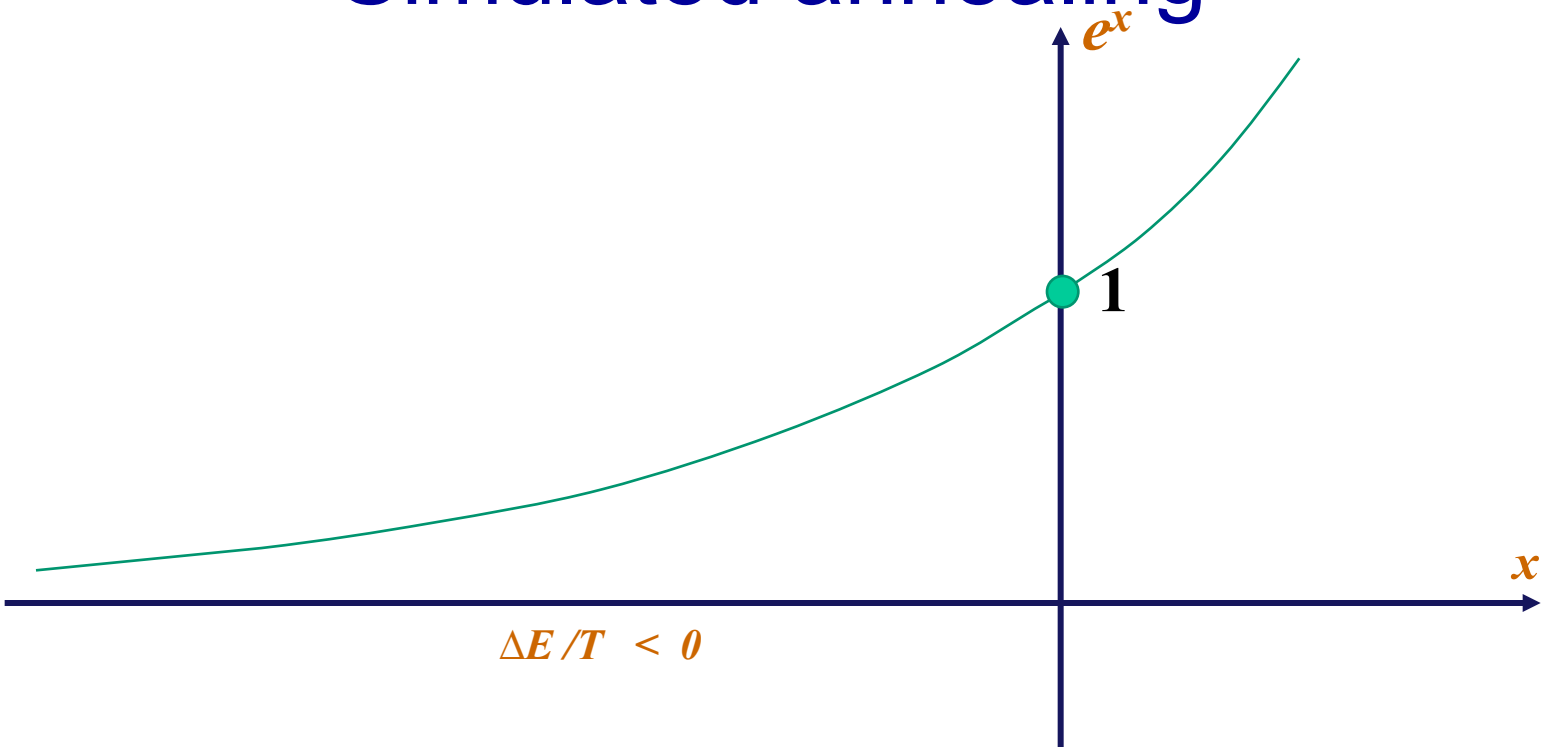
Δ*E* ← VALUE[*next*] - VALUE[*current*]

if Δ*E* > 0 **then** *current* ← *next*

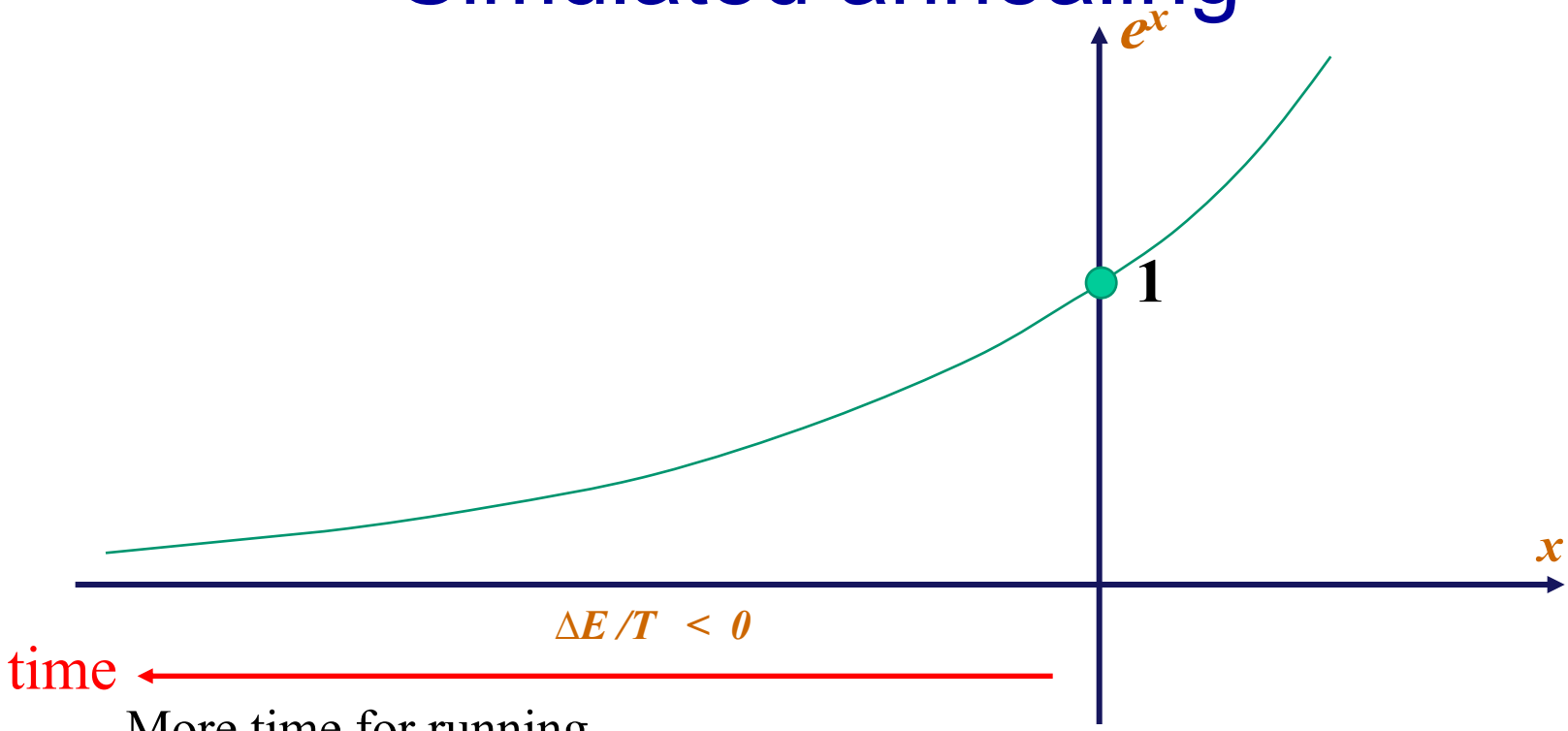
else *current* ← *next* only with probability $e^{\Delta E / T}$

Accept a state worse than the current

Simulated annealing



Simulated annealing



More time for running

$\Rightarrow$ Temperature (T): smaller and smaller

$\Rightarrow$ $|\Delta E/T|$: larger and larger

$\Rightarrow$ $\Delta E/T$: more negative

$\Rightarrow$ probability for accepting a worse state be smaller and smaller

Local beam search

Local beam search

- Keep track of *k* states instead of one
  - Initially: *k* random states
  - Next: determine all successors of *k* states
  - If any of successors is goal → finished
  - Else select *k* best from successors and repeat.
- Major difference with random-restart search
  - Information is shared among *k* search threads.
- Can suffer from lack of diversity.
  - Stochastic variant: choose k successors at proportionally to state success.

Local beam search

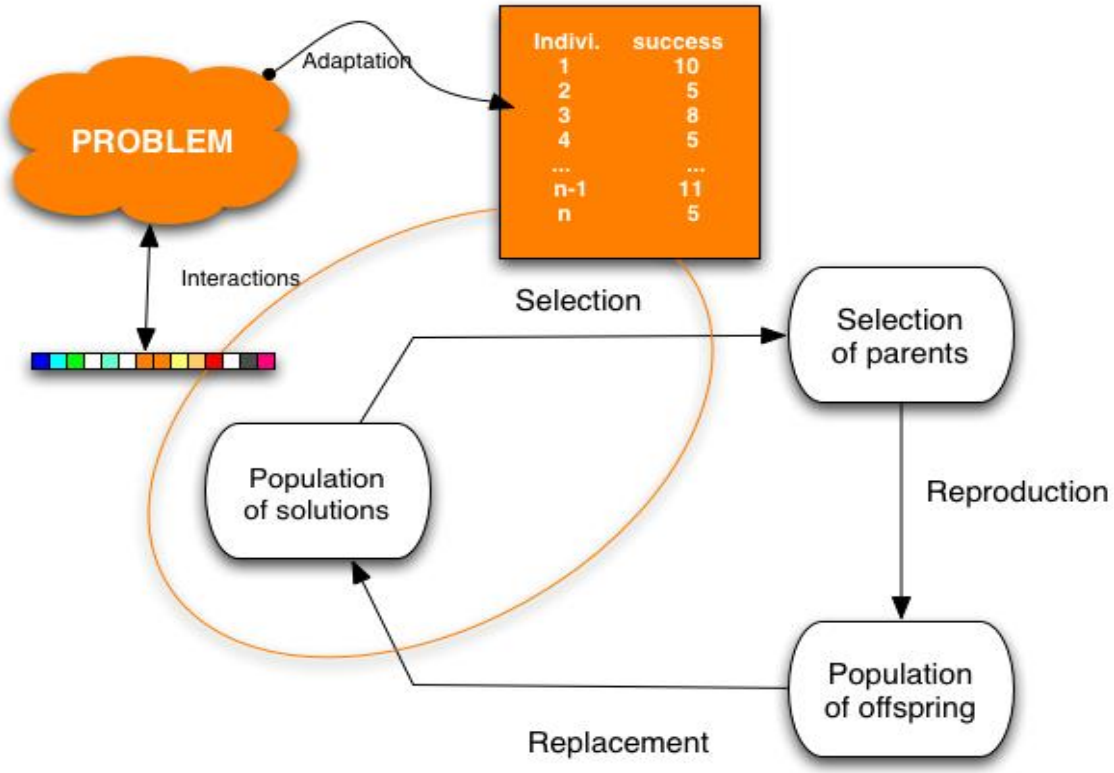
Different to running k instances of Hill-Climbing parallel

- Keep track of *k* states instead of one
  - Initially: *k* random states
  - Next: determine all successors of *k* states
  - If any of successors is goal → finished
  - Else select *k* best from successors and repeat.
- Major difference with random-restart search
  - Information is shared among *k* search threads.
- Can suffer from lack of diversity.
  - Stochastic variant: choose k successors at proportionally to state success.

Genetic algorithms

Genetic algorithms

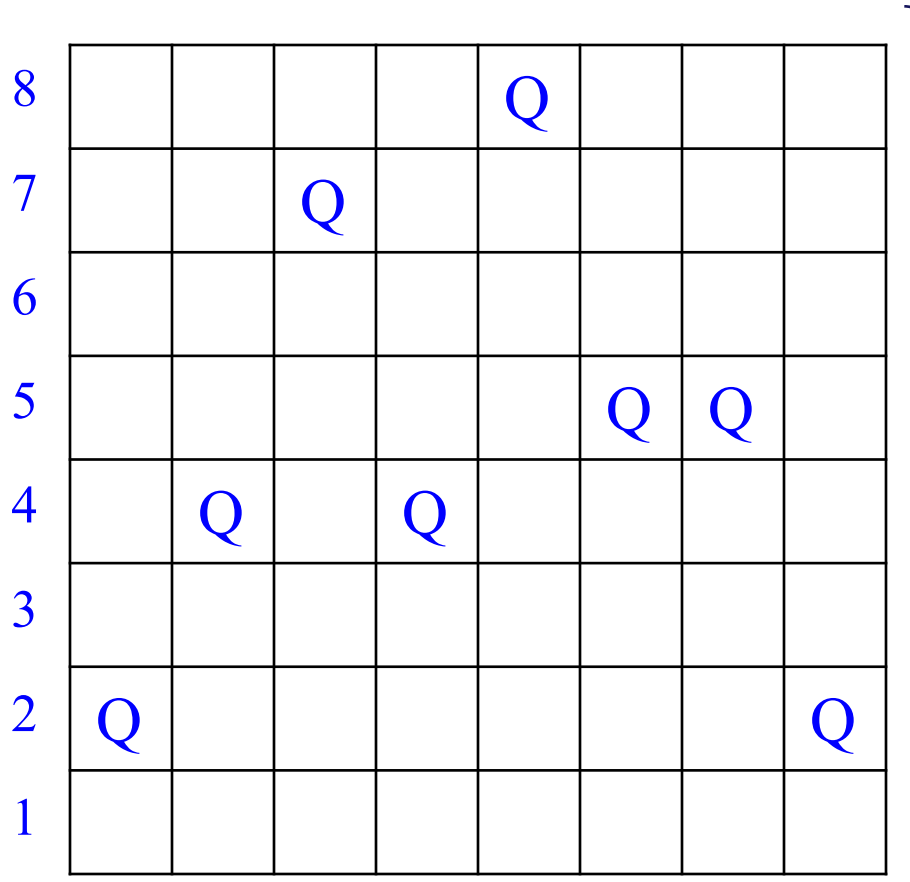
❖ Variant of local beam search with *sexual recombination*.



Genetic algorithms

- Tasks
  - Problem's states => genes: how to?
  - Genes: how to measure the goodness/fitness?
  - Mating individuals: how to?

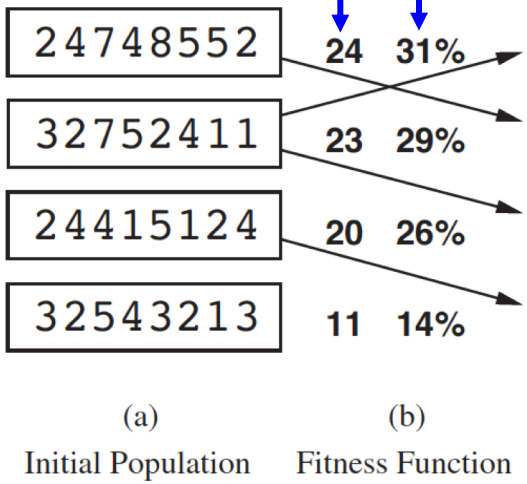
Genetic algorithms, example



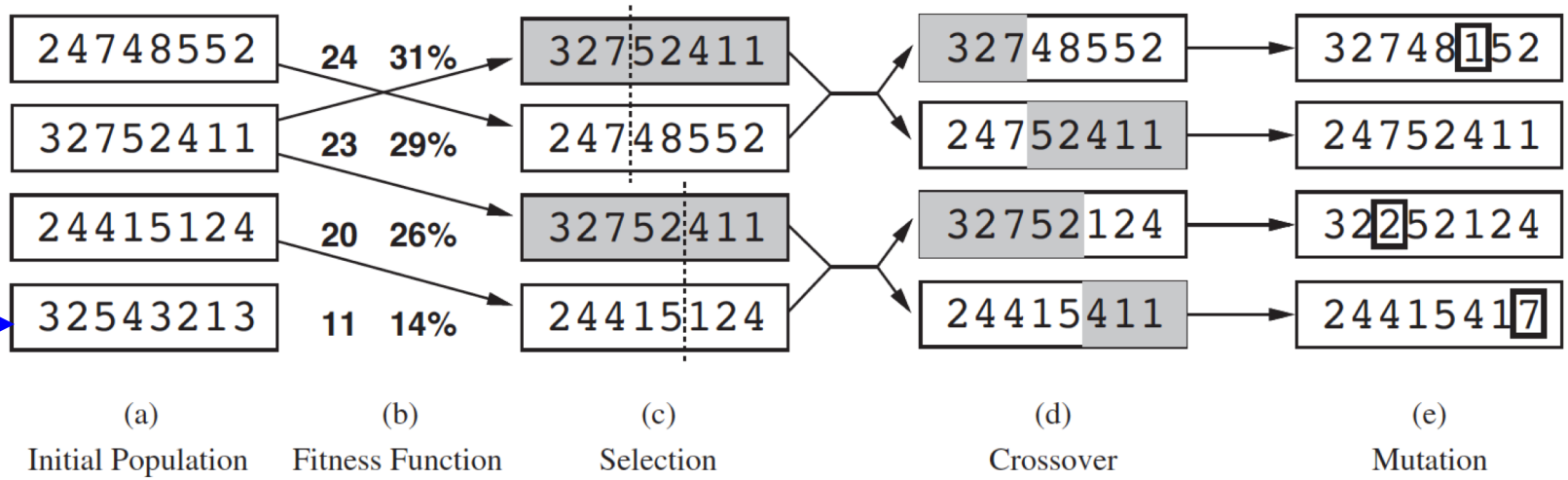
Percentage having matting:
~ fitness value

Fitness value:
#nonattacking pairs
(n(n-1)/2): max

Encoded
as a string

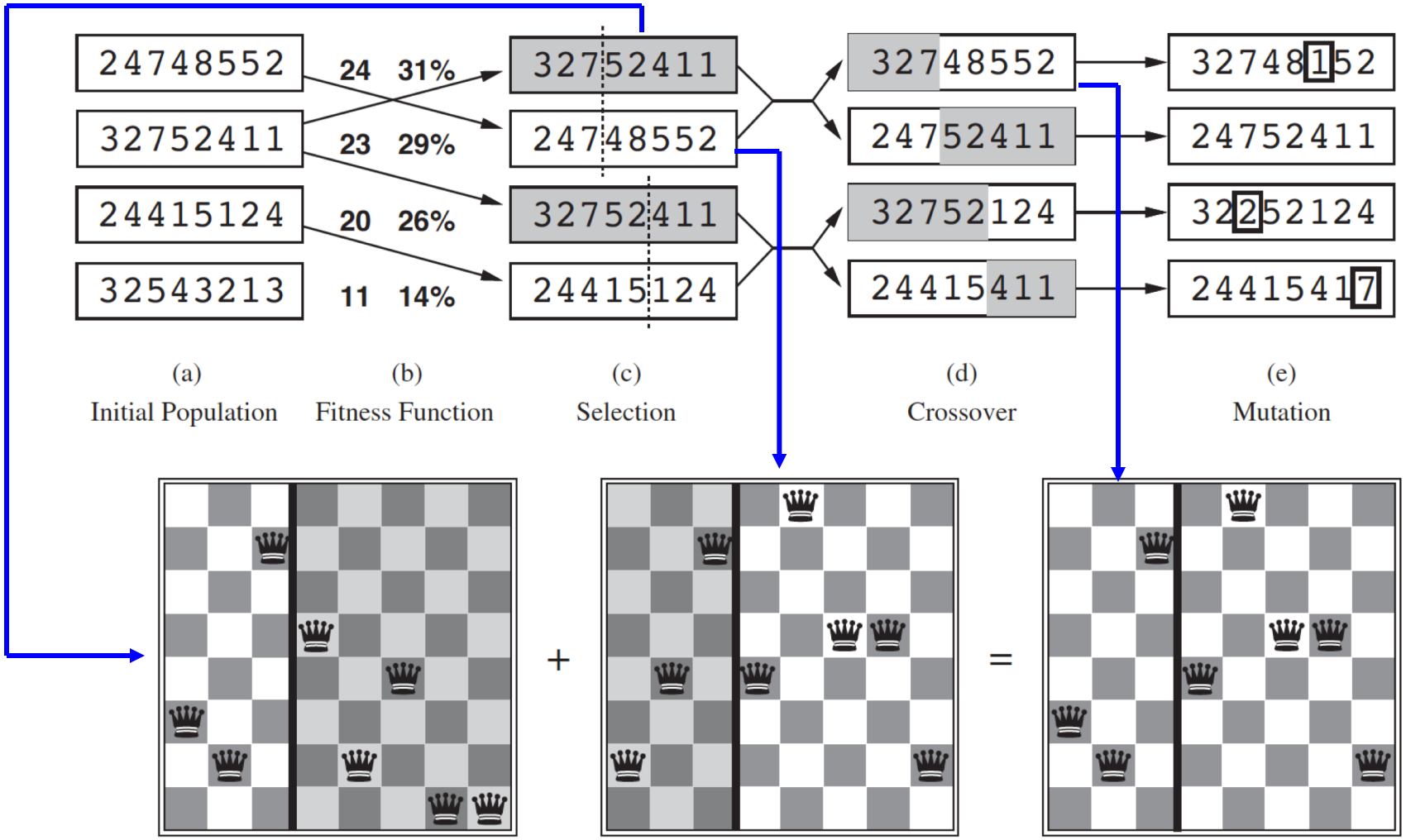


Genetic algorithms, example

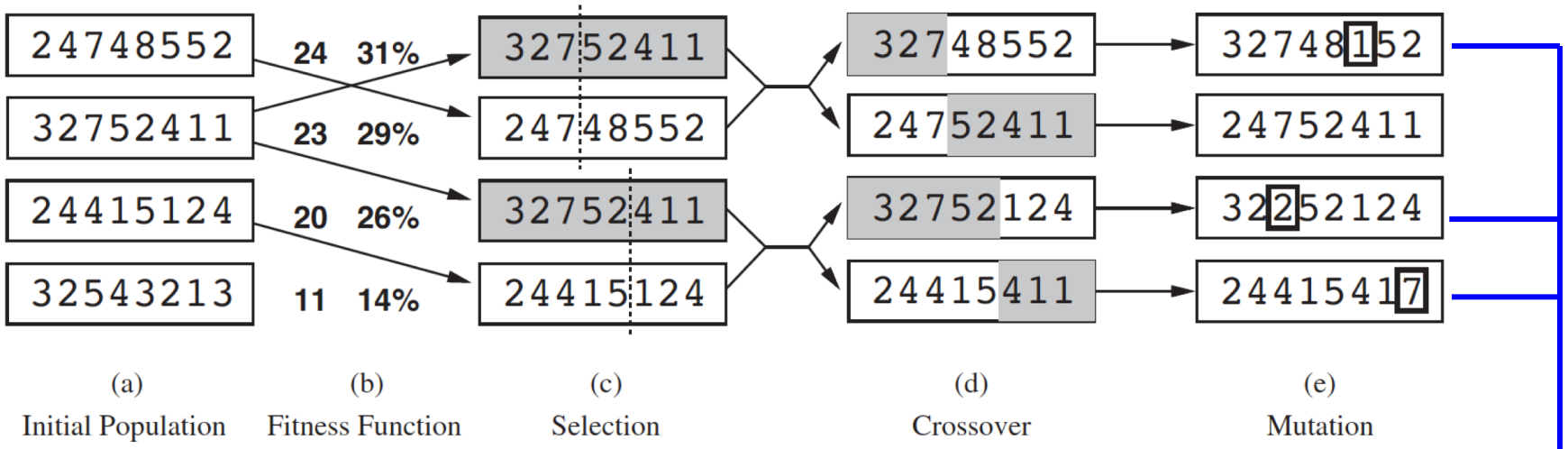


Some individuals may not have any chance for matting!!!

Genetic algorithms, example



Genetic algorithms, example



Have mutations with a small chance (probability)

Genetic algorithm

function GENETIC_ALGORITHM(*population,* FITNESS-FN) **return** an individual

input: *population*, a set of individuals

FITNESS-FN, a function which determines the quality of the individual

repeat

new_population ← empty set

loop for i **from** 1 **to** SIZE(*population*) **do**

x ← RANDOM_SELECTION(*population*, FITNESS_FN)

y ← RANDOM_SELECTION(*population*, FITNESS_FN)

child ← REPRODUCE(*x*,*y*)

if (small random probability) **then** *child* ← MUTATE(*child*)

add *child* to *new_population*

population ← *new_population*

until some individual is fit enough or enough time has elapsed

return the best individual