Ch4 Beyond Classical Search

- ✓ Hill-climbing search
- ✓ Simulated annealing
- ✓ Local beam search
- ✓ Genetic algorithms

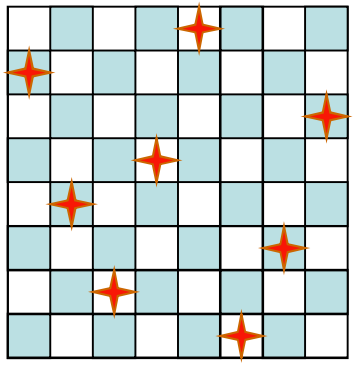
這一章的方法都不保證能找到最佳解！

Shun-Shii Lin
Department of Computer Science & Information Engineering
National Taiwan Normal University

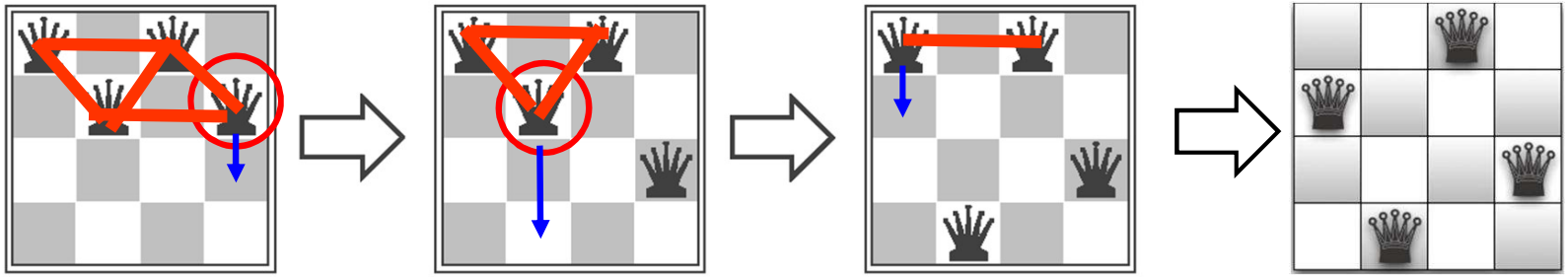
Acknowledgements: This presentation is created by Shun-Shii Lin based on the lecture slides from *The Artificial Intelligence: A Modern Approach by Russell & Norvig, a PowerPoint version by Berlin Chen,* and various materials from the web.

4.1 Local Search & Optimization

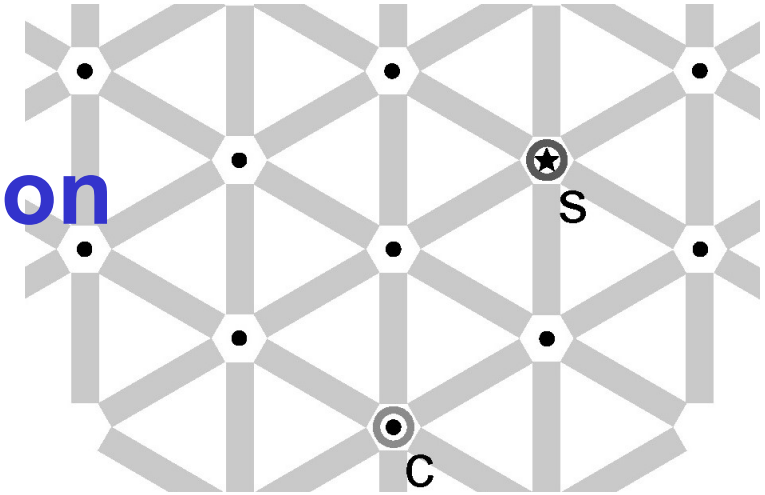
- In many optimization, path to solution is irrelevant.
  - e.g., 8-queen, VLSI layout, TSP etc., for finding optimal configuration.
  - The goal state itself is the solution.
  - The state space is a complete configuration.

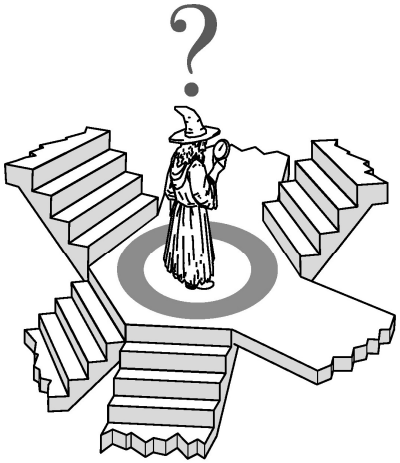


- In such case, local search algorithms can be used.
  - Start with a complete configuration, all queens are already placed, 1 per column.
  - Make modifications to improve the quality: moves 1 queen to another square within the same column.

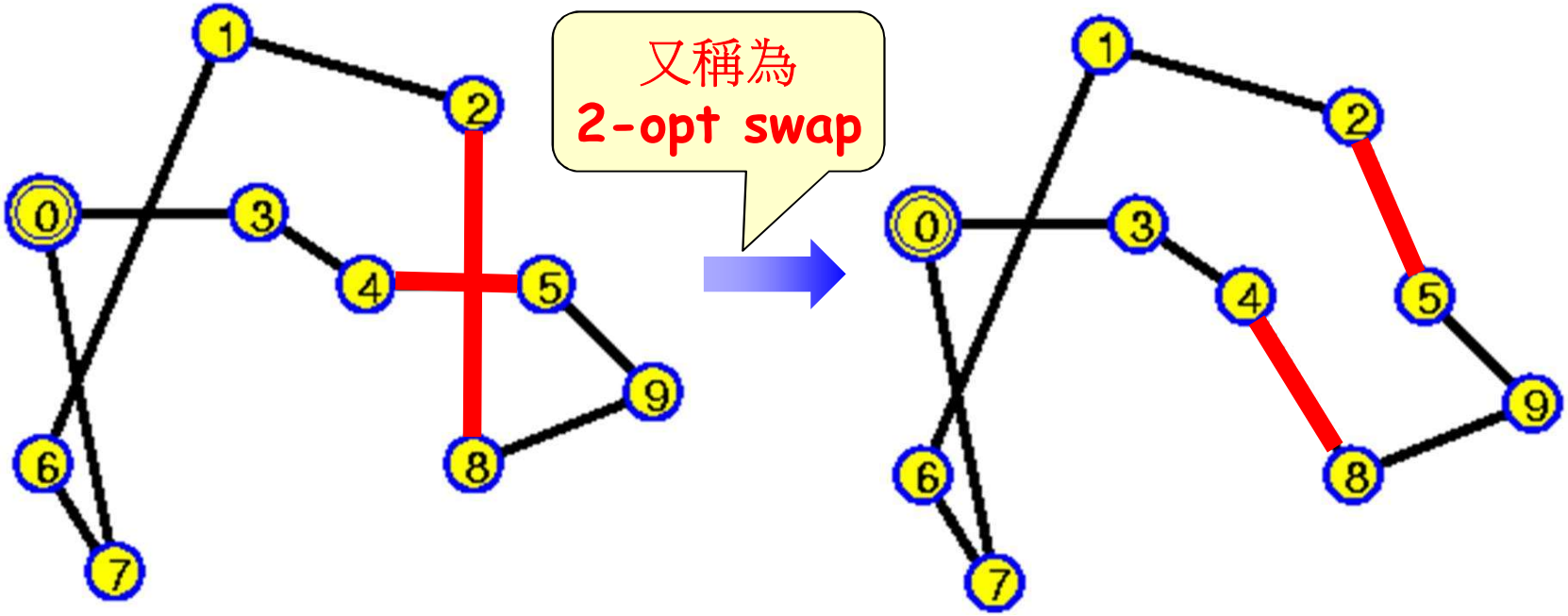


Local search: only use local information





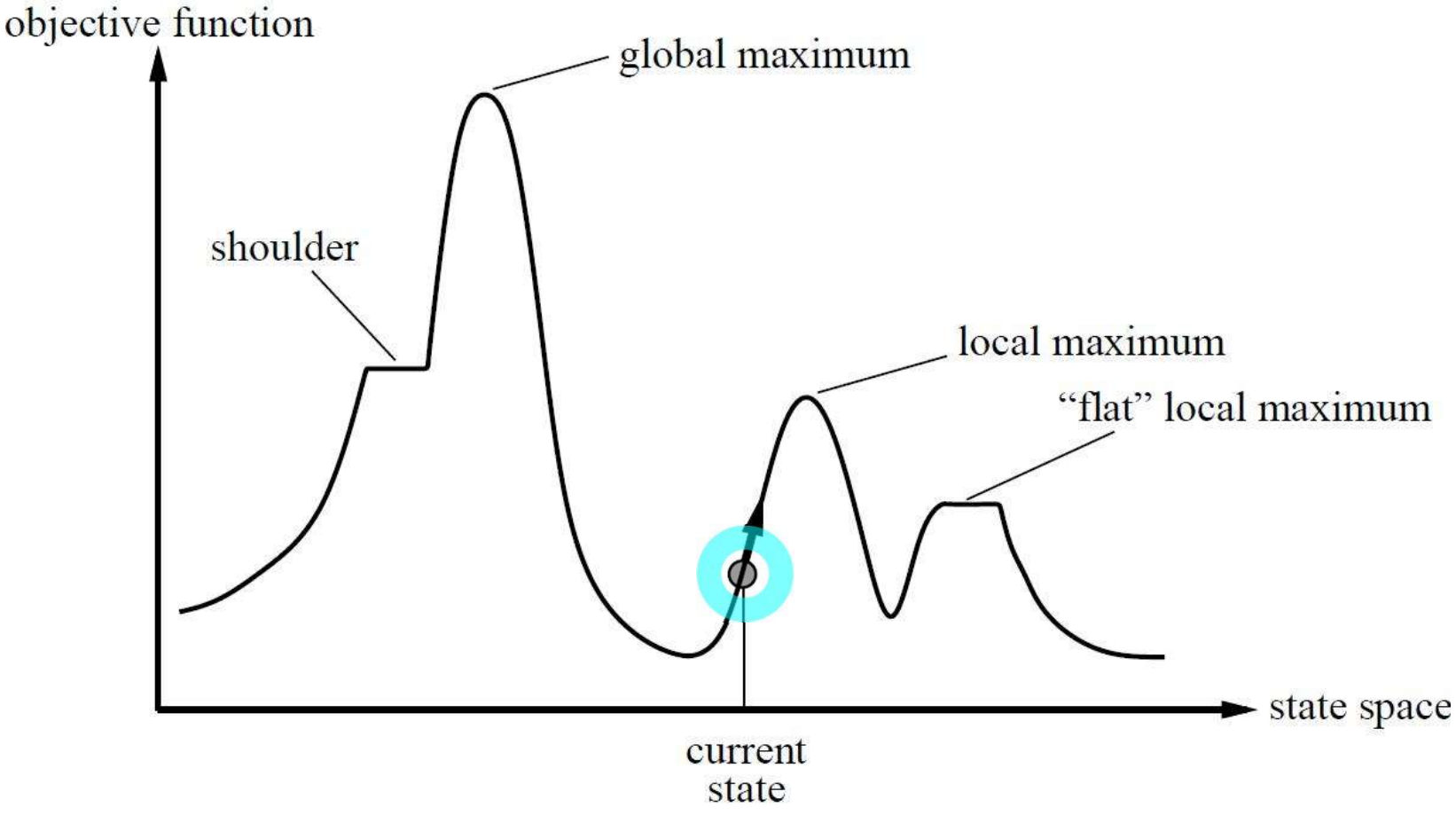
- Example: Traveling Salesperson Problem (TSP)
  - Find the shortest tour visiting all cities exactly once.
  - Start with any complete tour, perform pairwise exchanges if better.



Local search algorithms

4.1.1 Hill-climbing search 4.1.2 Simulated annealing
4.1.3 Local beam search 4.1.4 Genetic algorithms

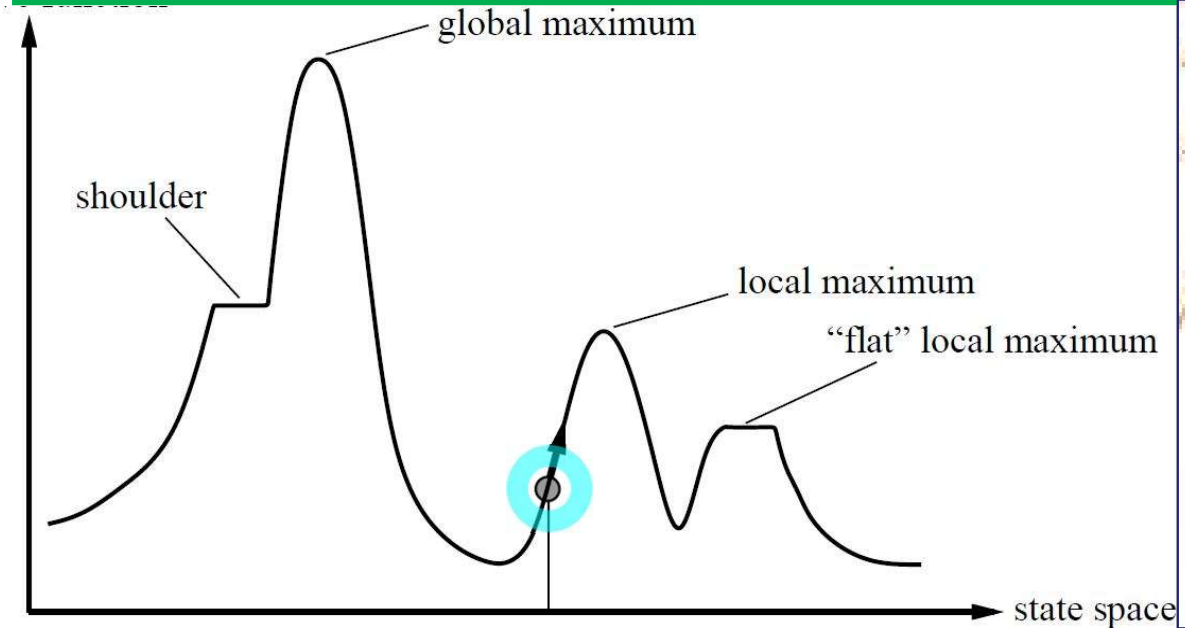
- *Local search algorithms* use a current state and generally move only to the neighbors of that state.
  - Use very little memory.

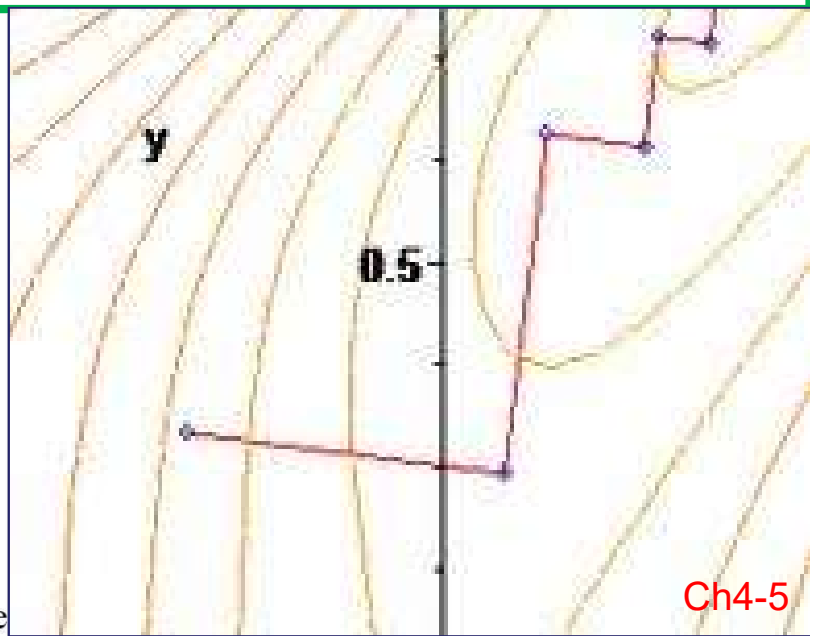


4.1.1 Hill-Climbing Search (*greedy local search*)

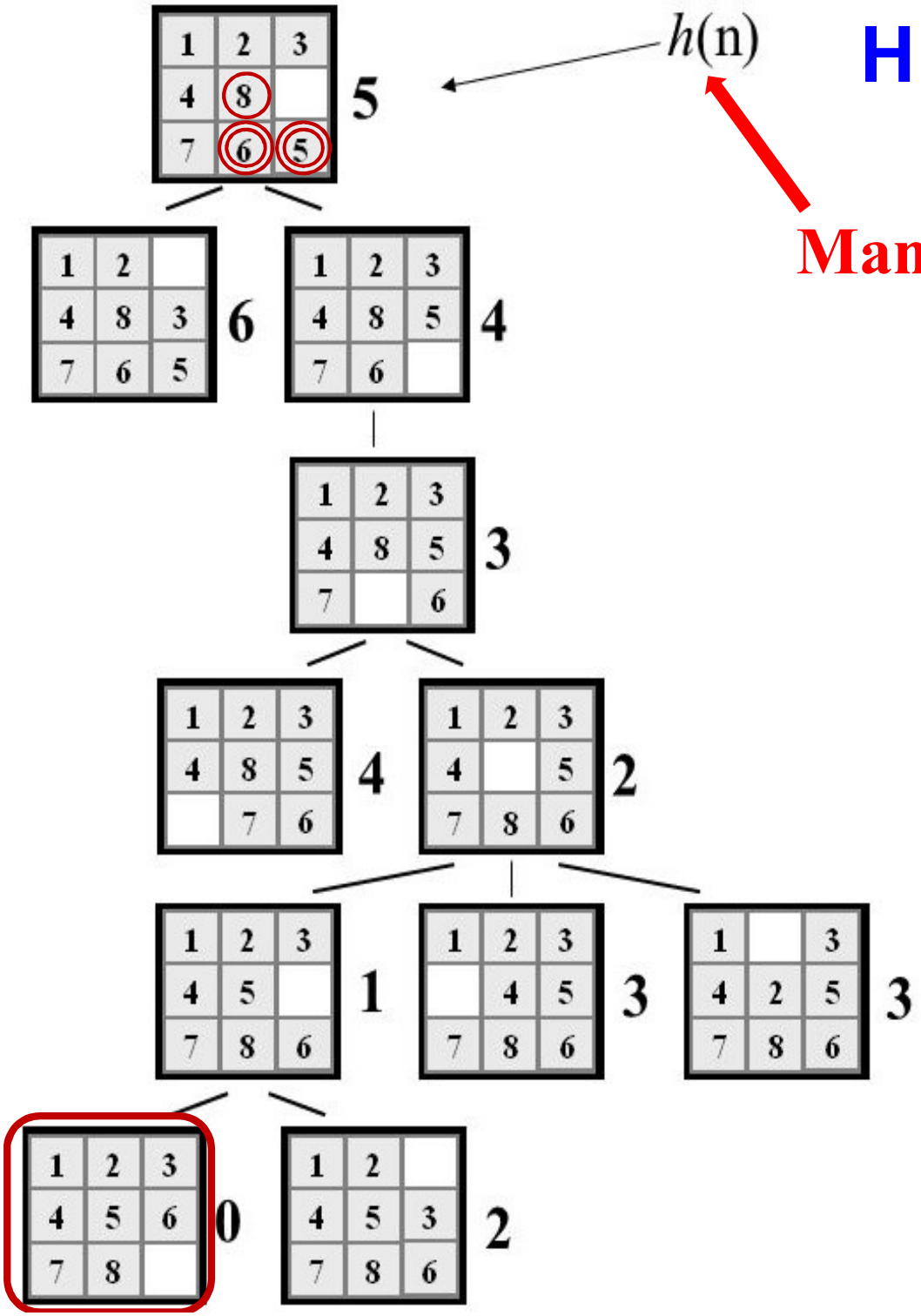
- "Like climbing Everest in the thick fog with amnesia."(健忘症)
- Choose any successor with a higher value (of objective or heuristic functions) than current state.
  - Choose *neighbor.*VALUE > *current.*VALUE

```
function HILL-CLIMBING(problem) return a state that is a local maximum
    current ← MAKE-NODE(problem.INITIAL-STATE)
    loop do
        neighbor ← a highest valued successor of current
        if neighbor.VALUE ≤ current.VALUE then return current.STATE
        current ← neighbor
```



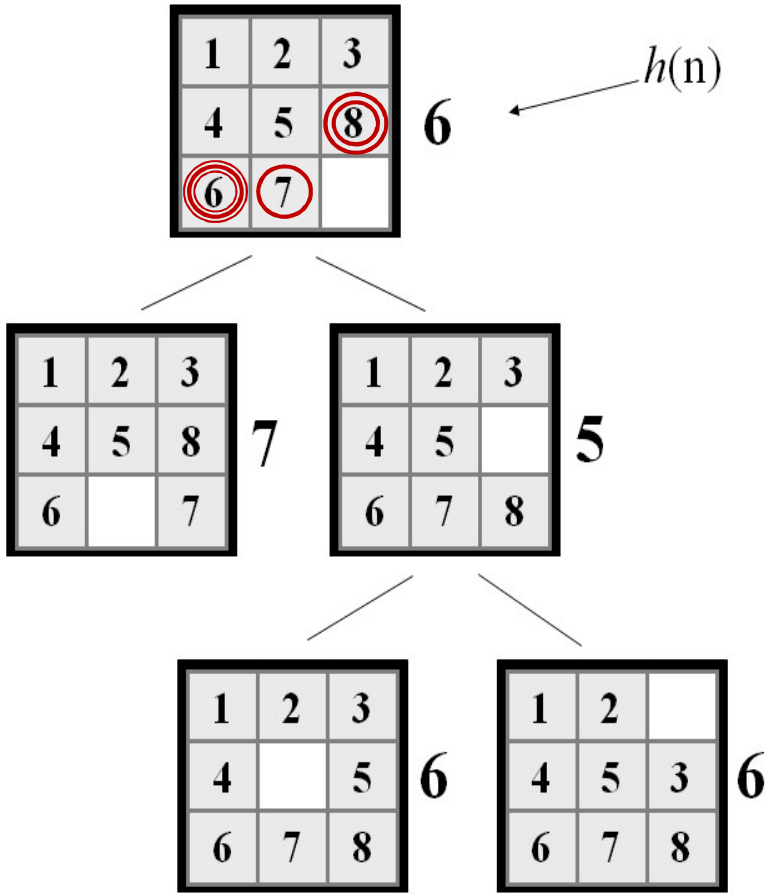


Hill climbing example: 8-puzzle



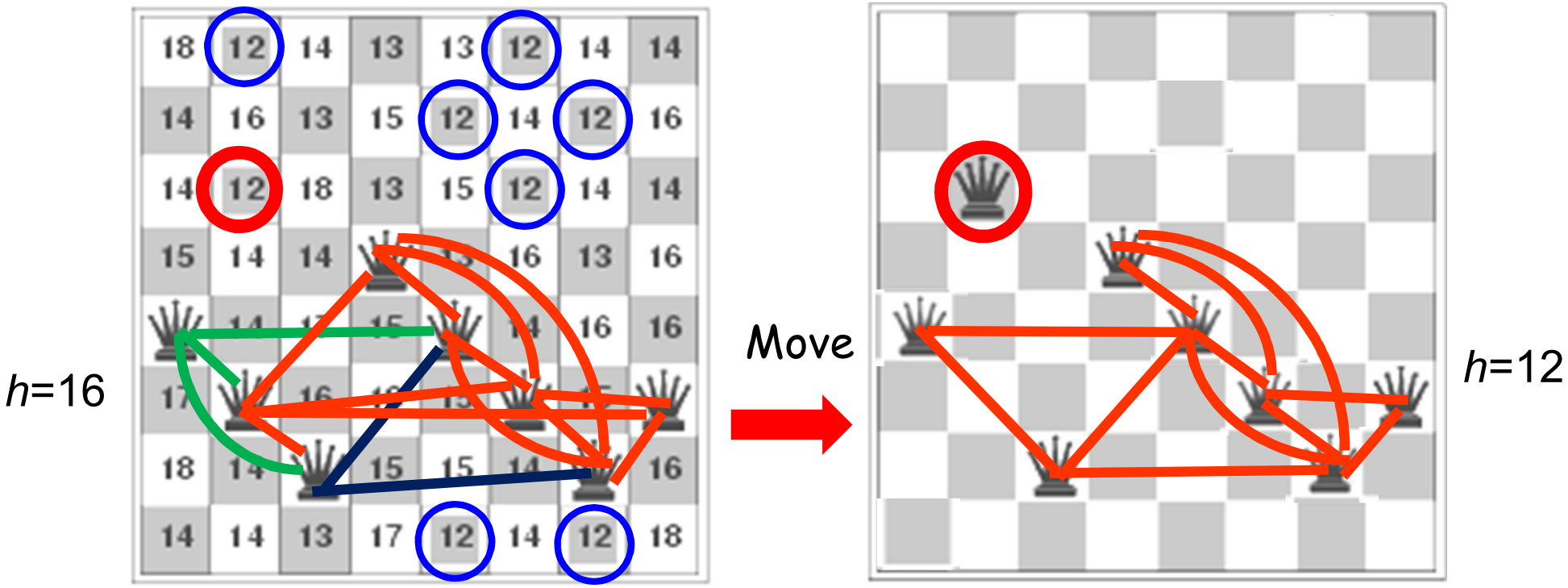
Manhattan distance heuristic

Example of a local optimum



Hill climbing example : the 8-queens problem

– The heuristic cost function is the number of pairs of queens that are attacking each other.



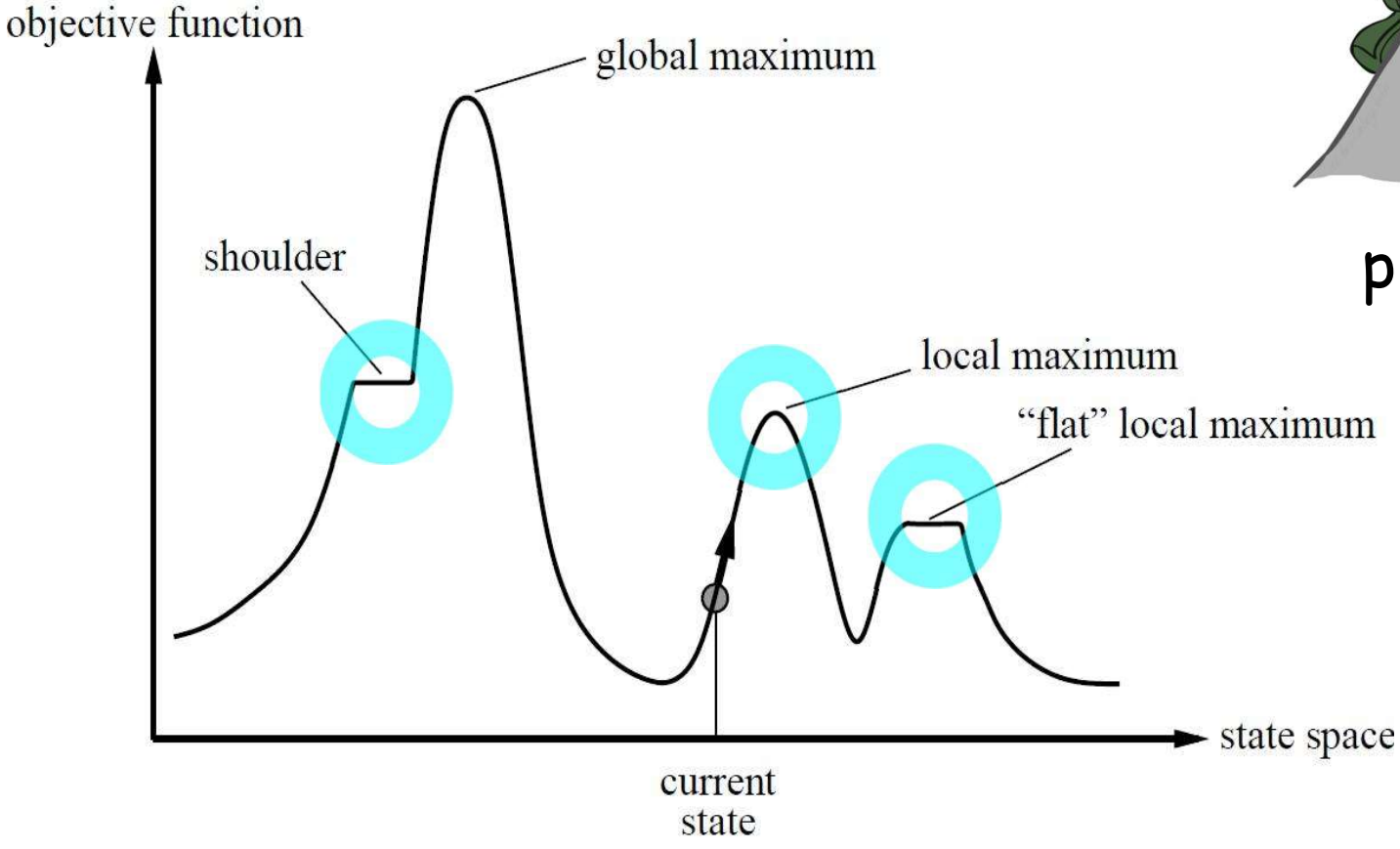
– h=3+4+2+3+2+2+1+0=16 (calculated from left to right.)

– Best successors have h=2+0+2+3+2+2+1+0=12. (when one queen in Column 2, 5, 6, and 7 is moved.)

– Hill-climbing typically chooses randomly among the set of best neighbors.

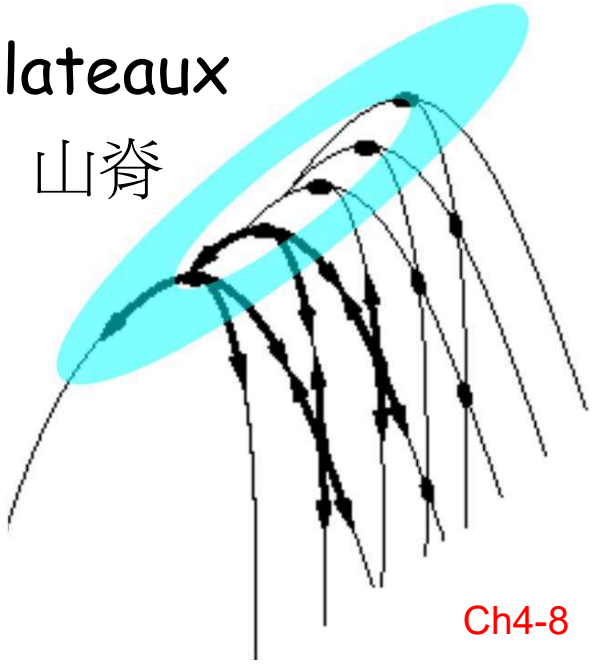
- Hill climbing is greedy and often makes very rapid progress towards a solution.
- Unfortunately, hill climbing often gets stuck for the following cases:
  - shoulder(ridges高原)
  - local maximum
  - flat local maximum(plateaux山脊)

  In each case, no progress can be made.

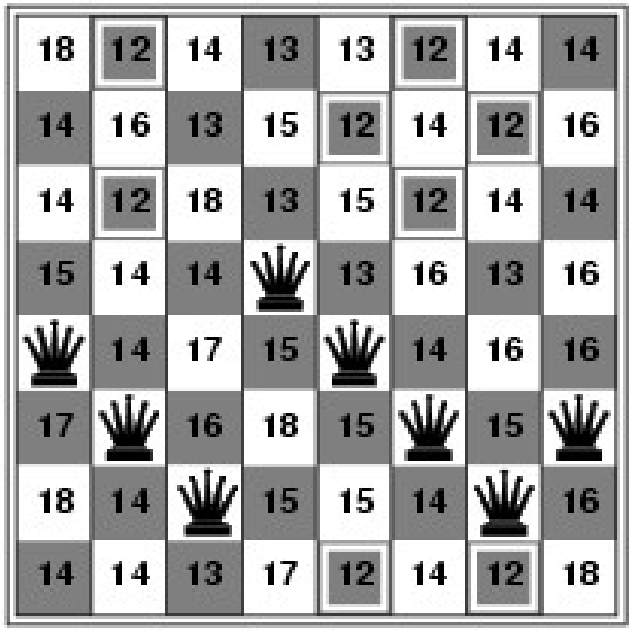


plateaux

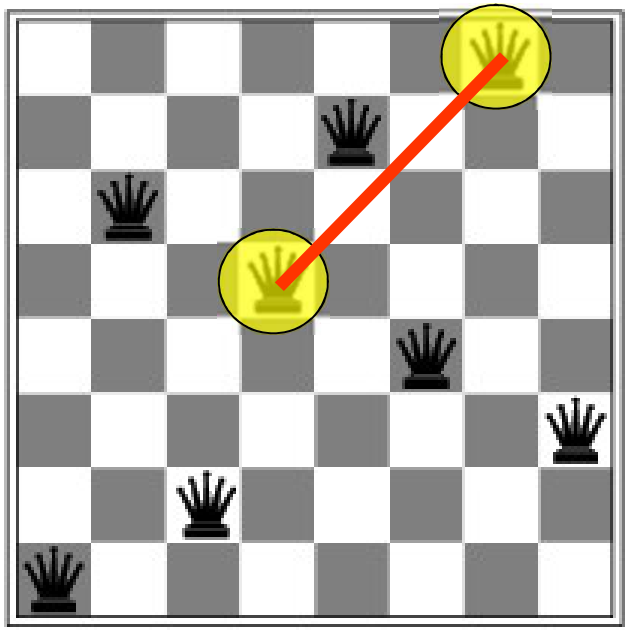
山脊



- Starting from a randomly generated 8-queens state, hill climbing gets stuck 86% of the time.

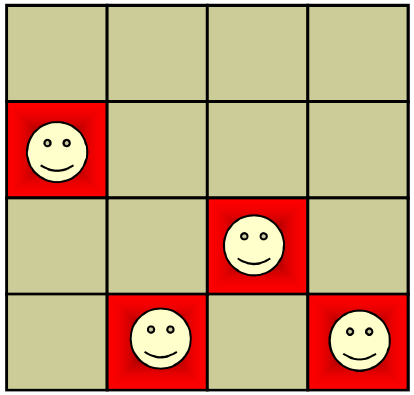


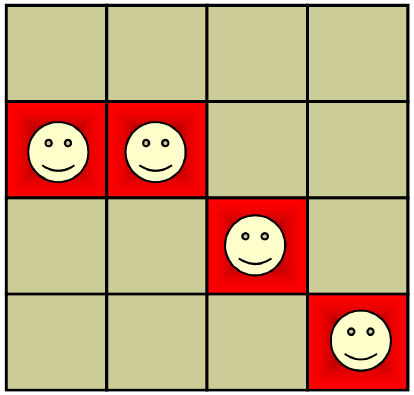
5 Moves



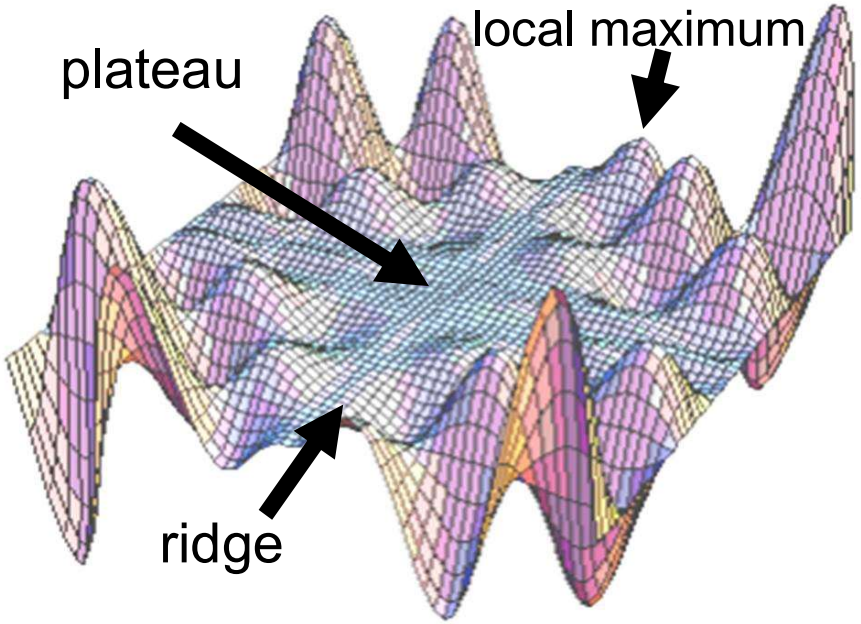
8-queens stuck in a local minimum(h=1)

- Exercise: Apply the hill-climbing search to solve the following two 4-queens problems.





Hill-Climbing Search



http://classes.yale.edu/fractals/CA/GA/Fitness/Fitness.html

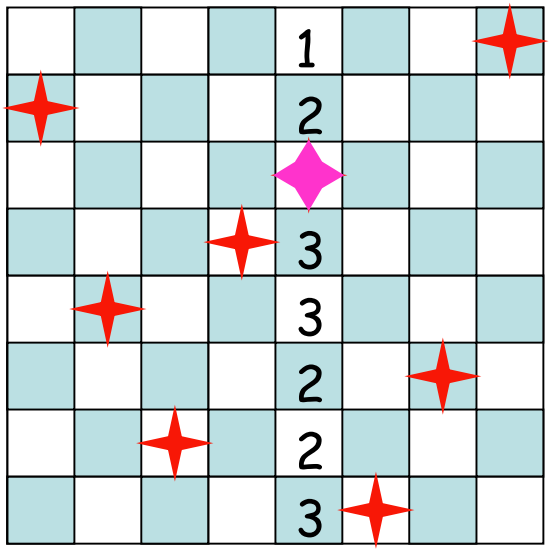
- Several variants:
  - Stochastic hill-climbing
    - Choose at random from among the *uphill* moves.
  - First-choice hill-climbing
    - Generate successors *randomly* until one that is better than current state is generated.
    - A kind of stochastic hill climbing.
  - Random-restart hill-climbing
    - Conduct a series of hill-climbing searches from randomly generated initial states.
    - Stop when the goal is found.

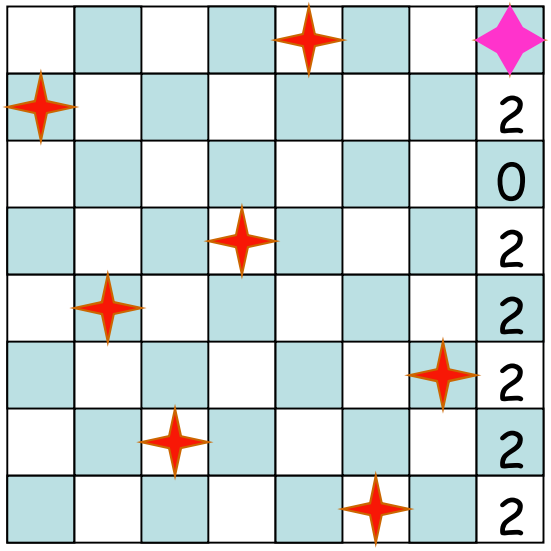
Random-Restart hill-climbing for 8-Queen

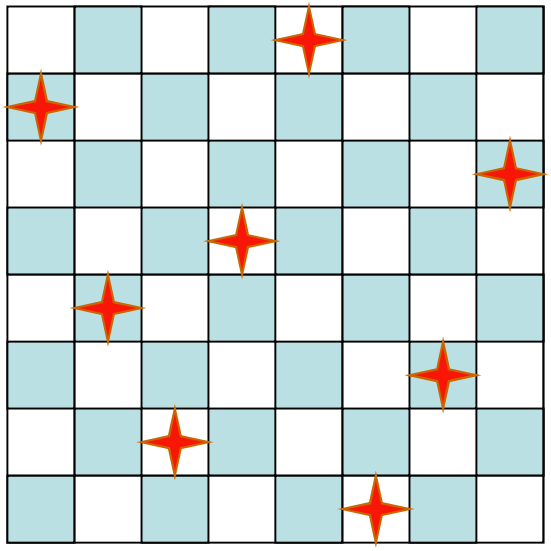
1. Repeat n times: // n = Max. number of restarts
   ① Pick an initial state S at random with one queen in each column.
   ② Repeat k times:
   - If GOAL?(S) then return S
   - Pick an attacked queen Q at random.
   - Move Q in its column to minimize the no. of attacking → new S
2. Return failure.

min-conflicts heuristic

DEMO Ch04-展示EightQueens.exe

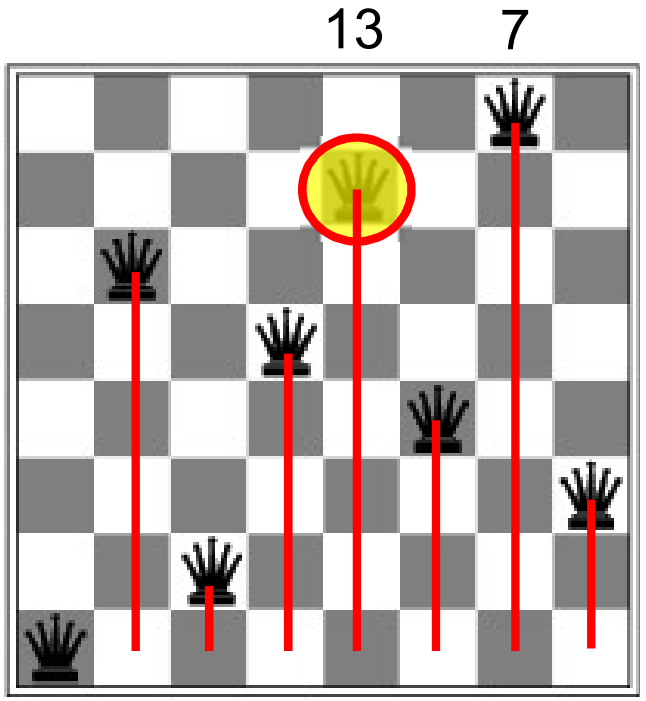






Why does it work ??? There are **many** goal states that are well-distributed over the state space. On the 3,000,000-queens problem, it typically finds a solution in < 1 minute.

- There could be different representations of solutions. The representation mechanism heavily affects the neighborhood function and the size of search space.



Representation scheme S1: $1 \le g[i] \le 64$

7	13	18	27	38	48	51	57

$64 \cdot 63 \cdot 62 \cdot \ldots \cdot 57 = 64!/56!$
$= 178{,}462{,}987{,}637{,}760$

Representation scheme S2: $1 \le g[i] \le 8$

7	5	2	4	6	8	3	1

$8 \cdot 8 \cdot 8 \cdot \ldots \cdot 8 = 8^8 = 16{,}777{,}216$

Representation scheme S3: $1 \le g[1] \le 8,\ 1 \le g[2] \le 7,\ \ldots\ 1 \le g[8] \le 1$

permutation of 1, 2, …, 8

7	5	2	**3**	**3**	**3**	**2**	**1**

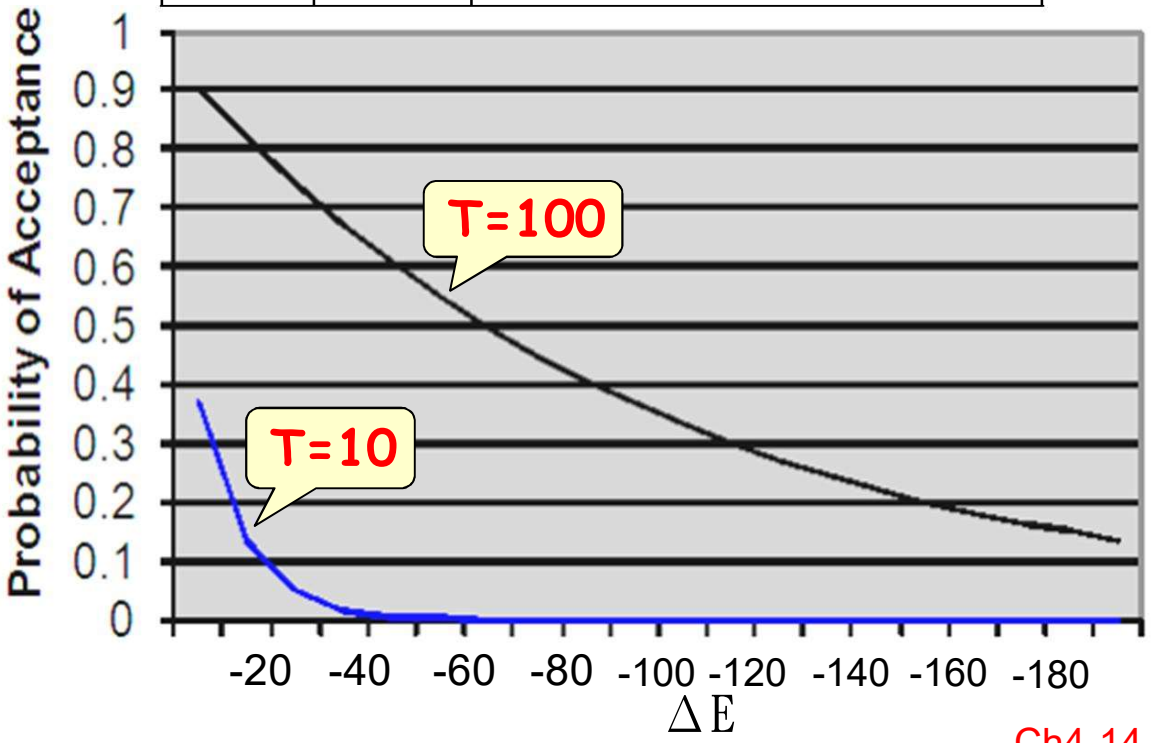
$8 \cdot 7 \cdot 6 \cdot \ldots \cdot 2 \cdot 1 = 8! = 40{,}320$

4.1.2 Simulated Annealing Search

- 模擬退火法是模擬冷卻晶體的過程。
- Metropolis等人在1953年提出。
- 1983年，Kirkpatrick等人將其運用在求最佳化的問題上。
- Combine hill climbing with a random walk to yield both efficiency and completeness.
  - Pick a random move at each iteration instead of picking the best move. 亂數找尋鄰近的點。
  - $\Delta E \leftarrow$ *next*.VALUE − *current*.VALUE
  - If the move improves the situation → accept! 若找到的點比舊點好，則取之。
  - Otherwise($\Delta E < 0$) , have a probability ($e^{\Delta E/T}$) to move to a worse state. 否則依照機率決定是否取之。
    - The probability decreases exponentially as ΔE decreases.
    - The probability decreases exponentially as T (temperature) goes down (as time goes by).

ΔE	T	$e^{\Delta E/T} = 2.718^{\Delta E/T}$
-10	100	0.904837418
-20	100	0.818730753
-30	100	0.740818221
-40	100	0.670320046
-50	100	0.606530660
-60	100	0.548811636
-70	100	0.496585304
-80	100	0.449328964
-90	100	0.406569660
-100	100	0.367879441
-110	100	0.332871084
-120	100	0.301194212
-130	100	0.272531793
-140	100	0.246596964
-150	100	0.223130160

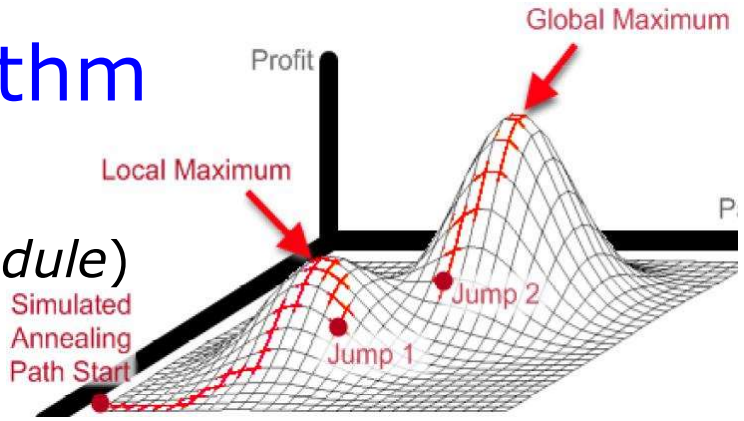
ΔE	T	$e^{\Delta E/T} = 2.718^{\Delta E/T}$
-10	10	0.367879441
-20	10	0.135335283
-30	10	0.049787068
-40	10	0.018315639
-50	10	0.006737947
-60	10	0.002478752
-70	10	0.000911882
-80	10	0.000335463
-90	10	0.000123410
-100	10	0.000045399



根據熱力學定律，在溫度為T的情況下，能量差所表現的機率=$P(\Delta E)=e^{\Delta E/kT}$
$= 2.718^{\Delta E/kT}$

k是Boltzmann Constant

Simulated Annealing Search algorithm



function SIMULATED-ANNEALING(*problem*, *schedule*)
returns a solution state
inputs: *problem*, a problem
schedule, a mapping from time to "temperature"

current ← MAKE-NODE(*problem*, INITIAL-STATE)　隨機產生一個初始解

for *t* = 1 **to** ∞ **do**　隨時間縮減溫度(冷卻過程) A typical cooling schedule is
T ← *schedule*(*t*)　*schedule*(*t*)= α· *schedule*(*t-1*),　(α約0.8~0.99為佳)

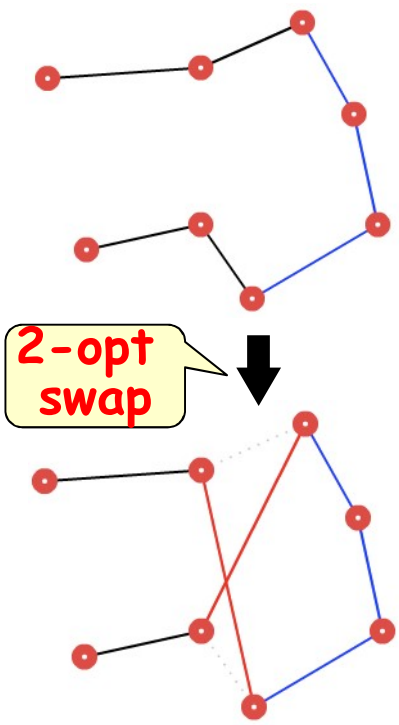
if *T* = 0 **then return** *current*　輸出最佳解
next ← a randomly selected successor of *current*　修改目前解，隨機挑一個successor

Δ*E* ← *next*.VALUE – *current*.VALUE
if Δ*E* > 0 **then** *current* ← *next*
else *current* ← *next* only with probability $e^{\Delta E/T}$　Be negative here!　是否接受?

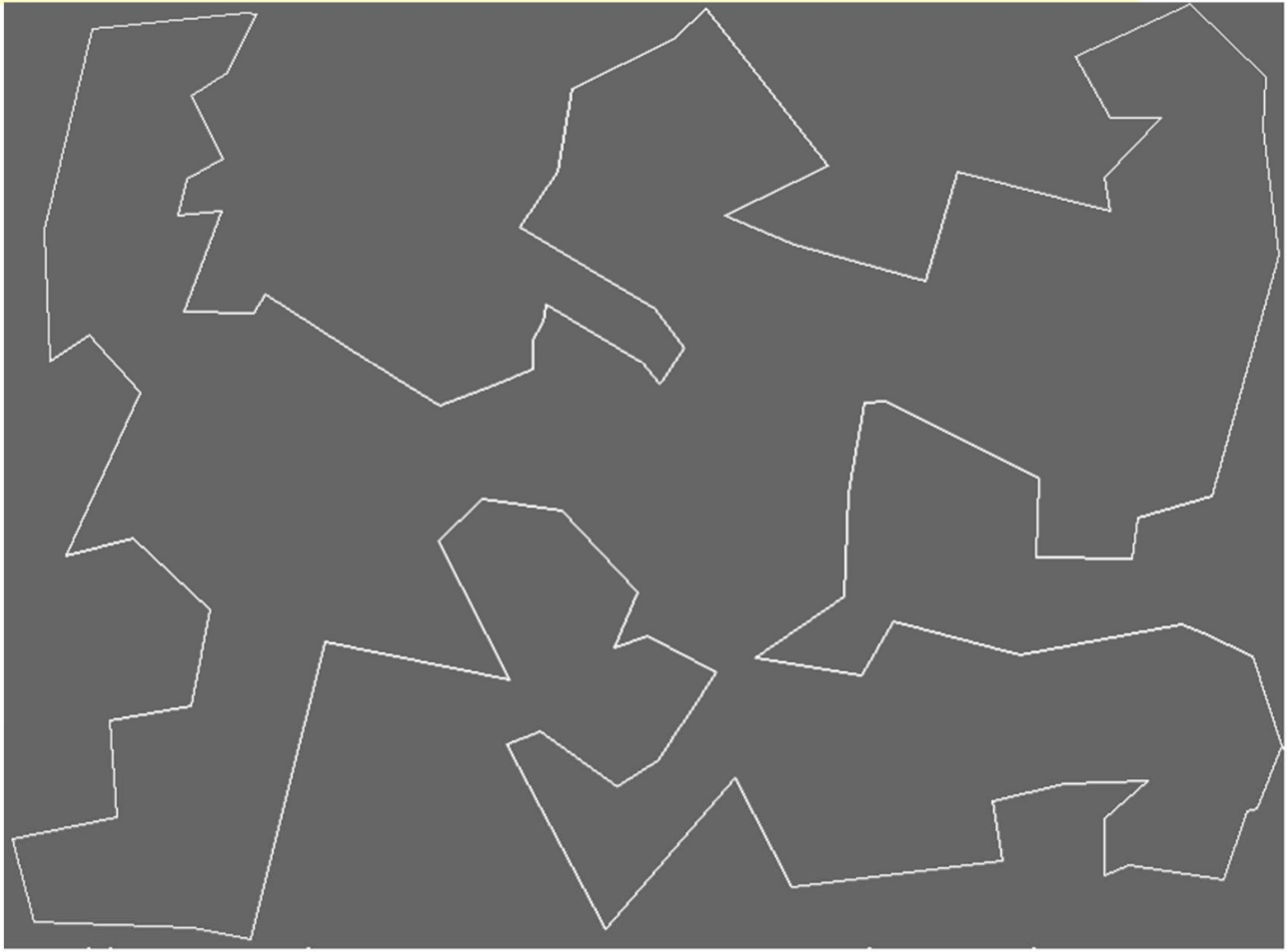
/* T is larger earlier, so probability of a worse move
declines over time */

Travelling Salesman with Simulated Annealing

http://www.staff.science.uu.nl/~beuke106/anneal/anneal.html



with some probability!

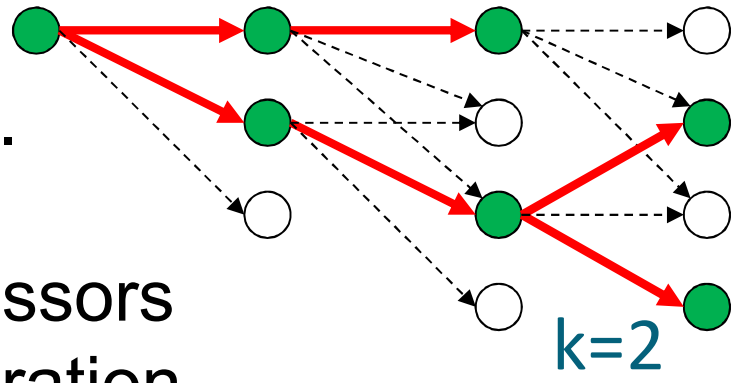


DEMO 2-OPT applied to TSP

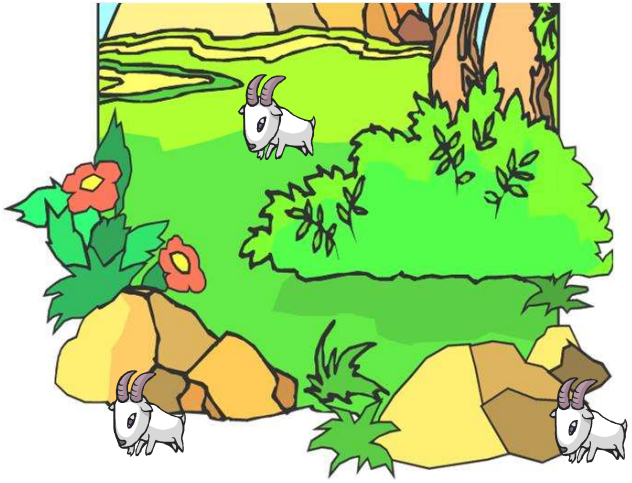
http://pedrohfsd.com/2017/08/11/2opt-part2.html

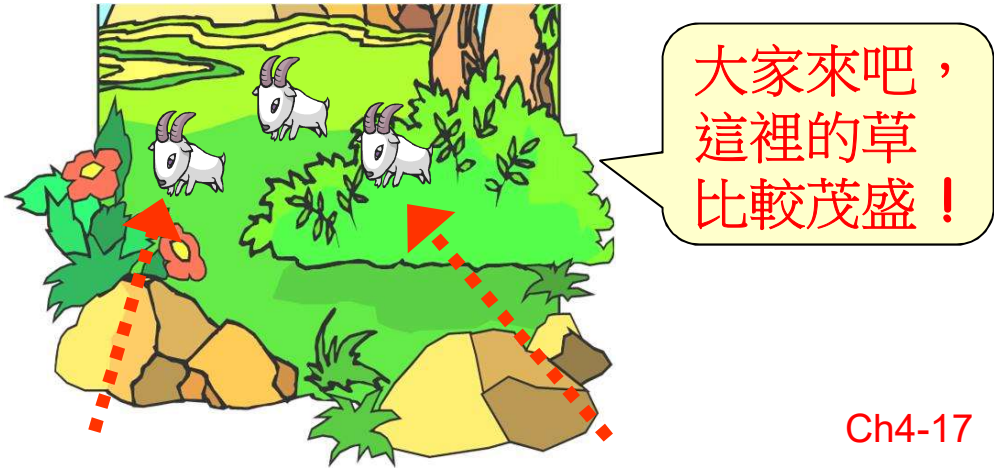
4.1.3 Local Beam Search(局部剪枝搜尋)

- Keep track of *k* states rather than just one.
  - Begin with *k* randomly generated states.
  - All successors of the *k* states are generated at each iteration.
    - If any one is a goal → halt!
    - Otherwise, select *k* best successors from them and continue the iteration.

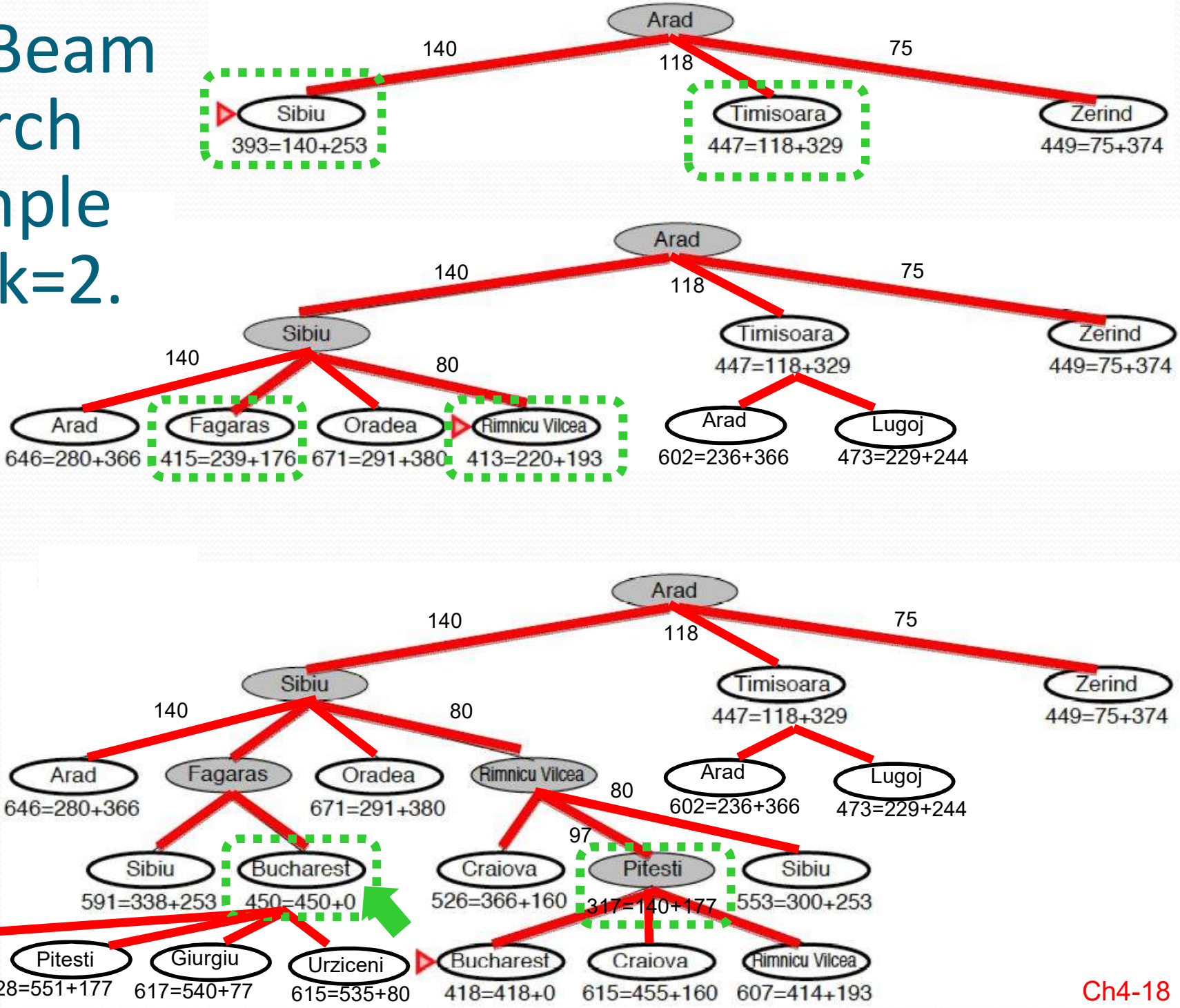


  - Information is exchanged among these *k* states.
    - Compared to the random-restart search.
      - Each process run independently.





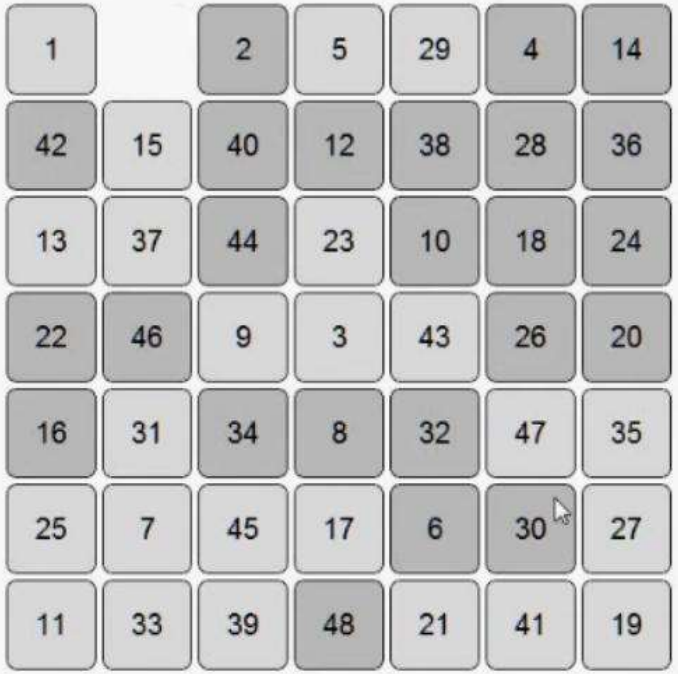
Local Beam Search Example with k=2.



Local Beam Search algorithm

```
function BEAM-SEARCH( problem, k) returns a solution state
    start with k randomly generated states
    loop
        generate all successors of all k states
        if any of them is a solution then return it
        else select the k best successors
```

- In 48-Puzzle(7×7 tile puzzle), A* may run out of memory since the space requirements can go up to order of 10^{61}.
- Experiment conducted shows that beam search with k=10,000 solves about 80% of random problem instances of the 48-Puzzle (7×7 tile puzzle).



DEMO Ch04-展示Puzzle15A.exe

Local Beam Search (cont.)

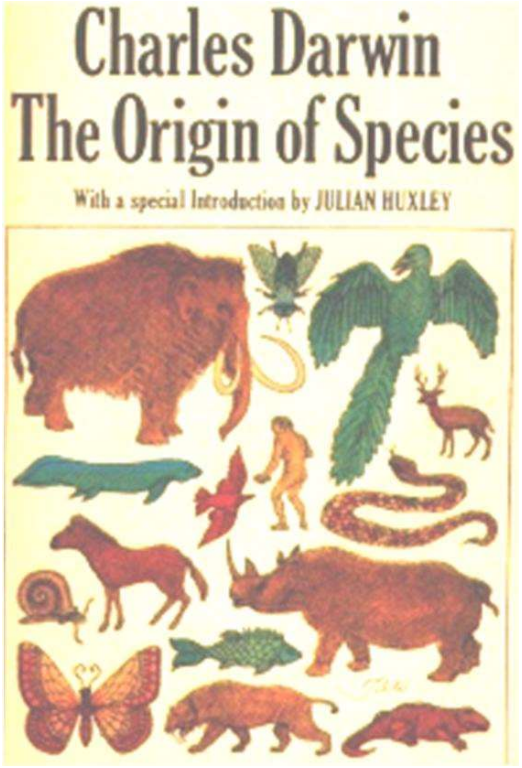
- Problem
  - The k states may quickly become concentrated in a small region of the state space.
  - Like an expensive version of hill climbing.

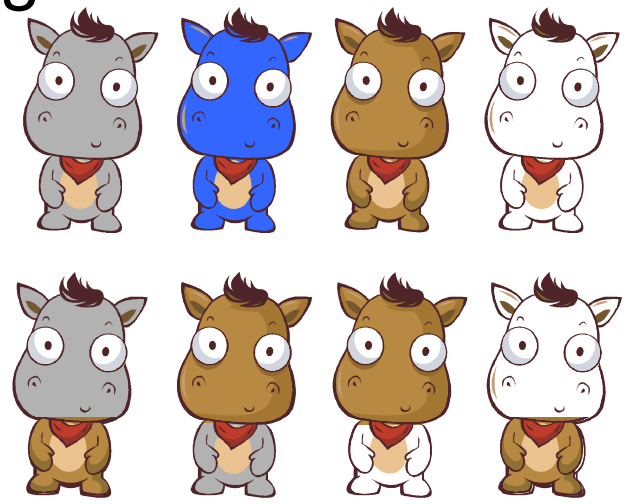
- Solution
  - A variant version called **stochastic beam search (統計式剪枝搜尋)**.
    - Choose a given successor at random with a probability in increasing function of its value.
    - Resemble the process of natural selection.

從所有successor中用亂數挑出 k 個，比較好的successor被挑中的機率就要比較差的successor高。

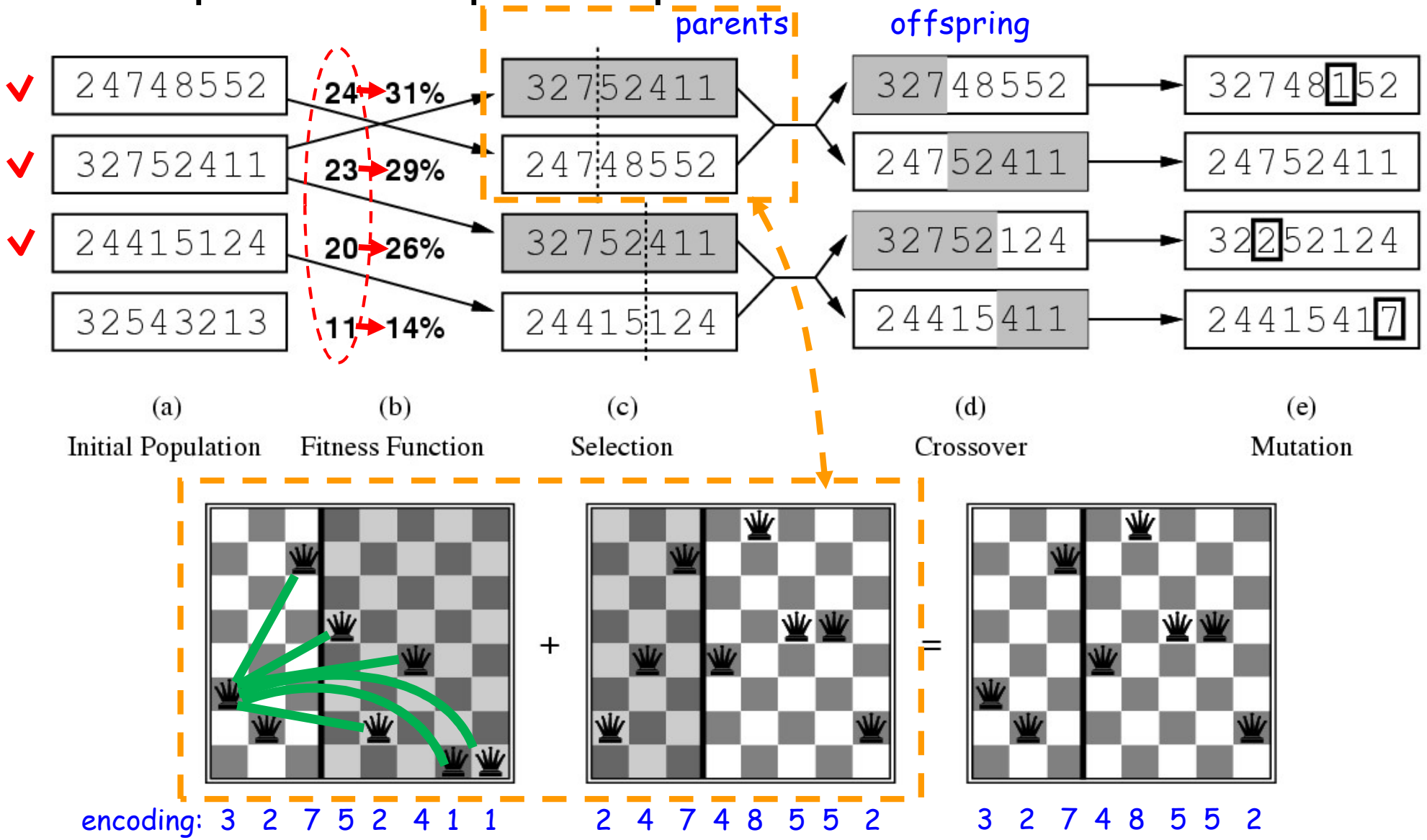
4.1.4 Genetic Algorithms (GA，遺傳演算法)

- Developed and patterned after biological evolution.
- Formally introduced in the US in the 70s by John Holland.
- Also regarded as a variant of stochastic beam search.
  - Successors are generated from multiple current states.
    - A population of potential solutions are maintained.
  - States are often described by bit strings (like chromosomes染色體) whose interpretation depends on the applications.
    - alphabet or binary-coded
      (11, 6, 9) → (1011011011001001)
  - Search begins with a population of randomly generated initial states.





- Example: the 8-queens problem



Fitness function = number of non-attacking pairs of queens=6+5+4+3+3+2+0+0=23

解的值最高為28=$\binom{8}{2}$

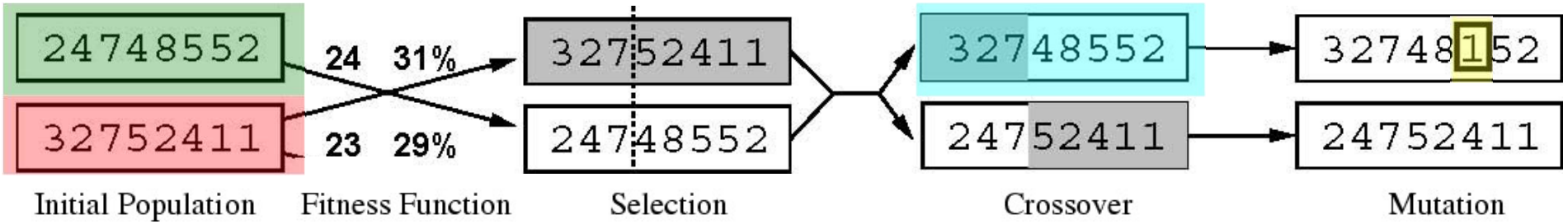
24/(24+23+20+11)=31%

23/(24+23+20+11)=29%

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{P} Fitness(h_j)}$$

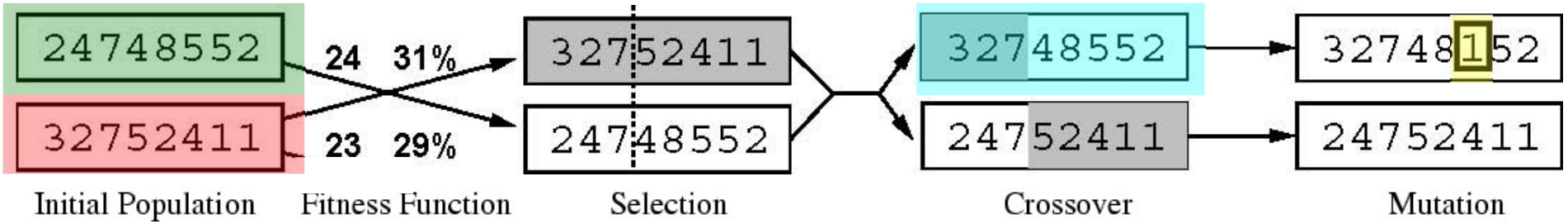
Genetic Algorithms (cont.)

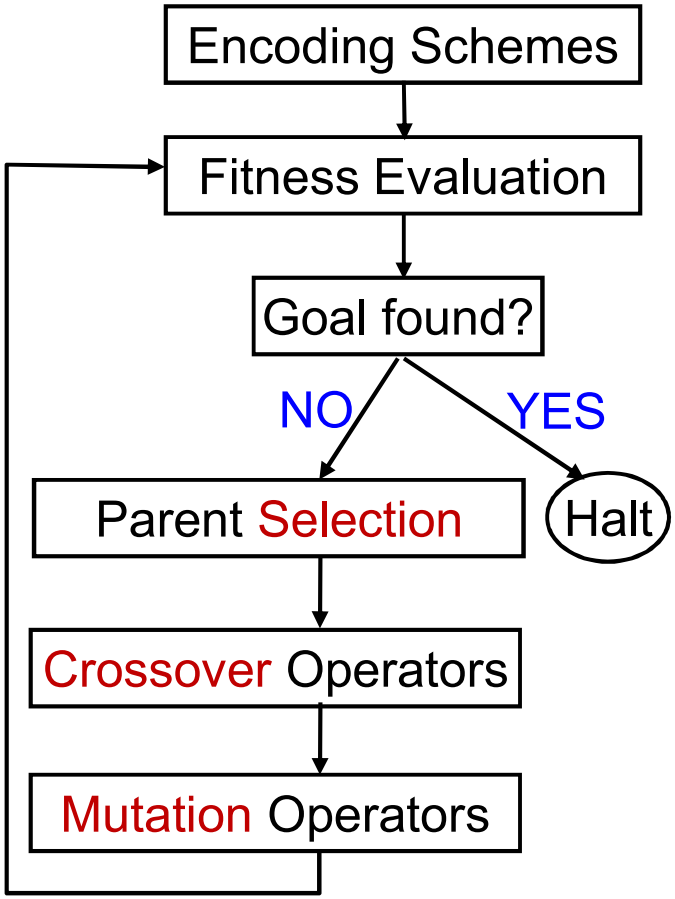
- The successor states are generated by combining two parent states, rather than by modifying a single state.



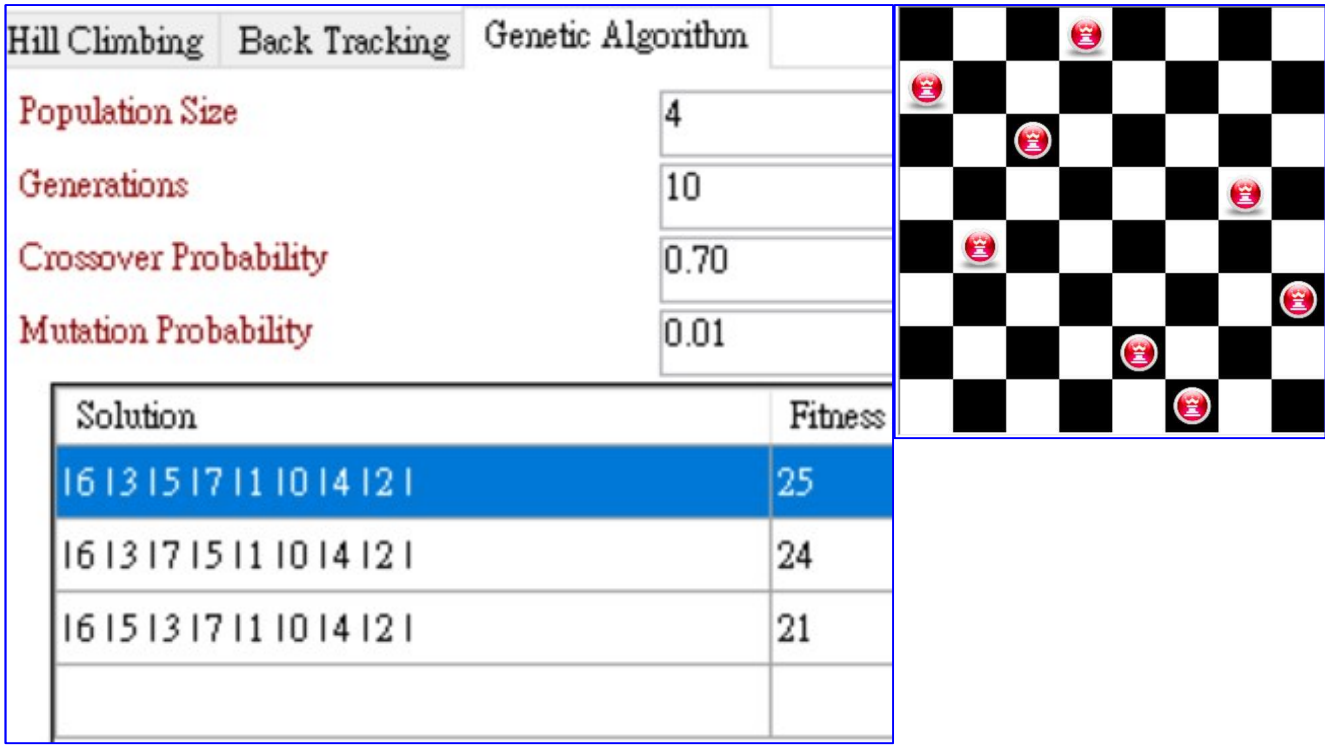
  - Current population/states are evaluated with a *fitness function* and selected probabilistically as seeds for producing the next generation.
    - Fitness function: the criteria for ranking.
    - Recombine parts of the best (most fit) currently known states.
  - Each (bit) location of the *randomly selected offspring* is subject to *random* **mutation** with a small independent probability.

- Three phases of GAs:
  Selection → Crossover → Mutation

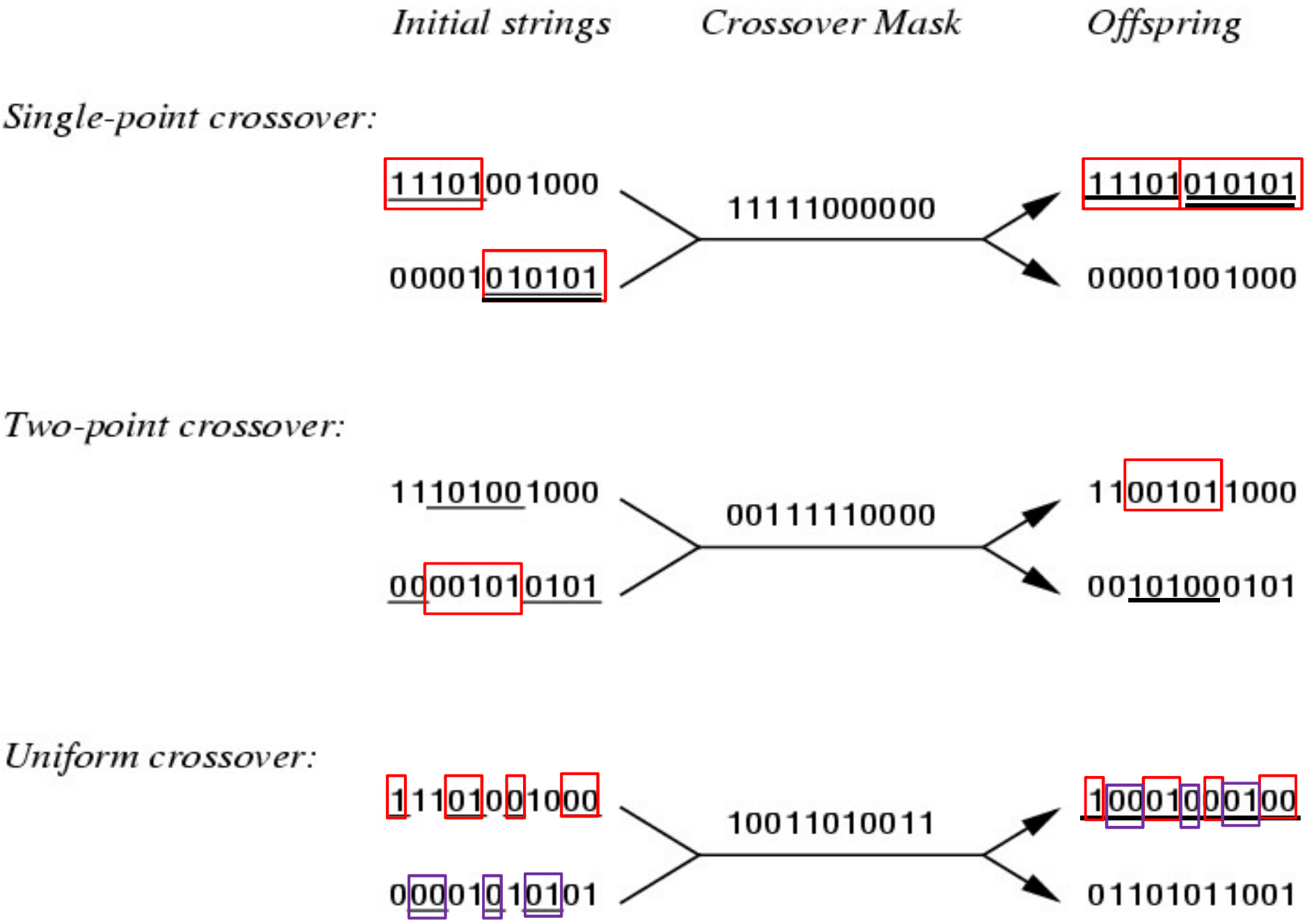




DEMO **Ch04-展示EightQueens.exe**



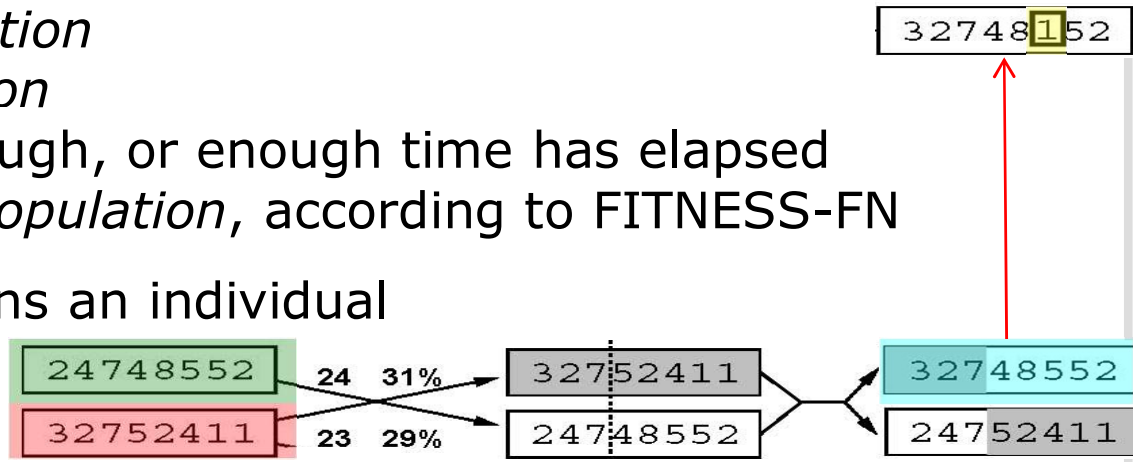
- Example: common crossover operators



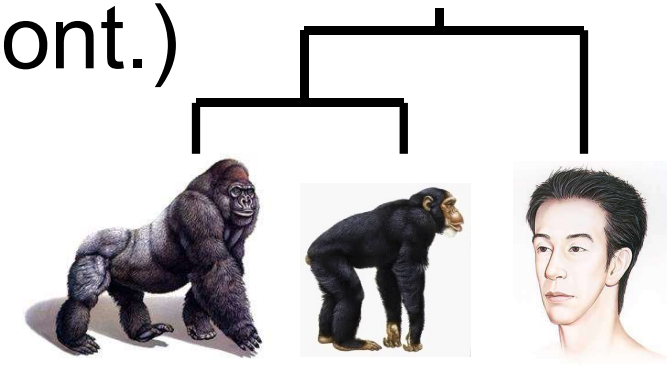
Genetic Algorithms

function GENETIC-ALGORITHM(*population,* FITNESS-FN) **returns** an individual
  inputs: *population*, a set of individuals
    FITNESS-FN, a function measuring fitness of an individual
  repeat
    new_population ← empty set
    for *i* = 1 **to** SIZE(*population*) **do**
      x ← RANDOM-SELECTION(*population*, FITNES-FN)
      y ← RANDOM-SELECTION(*population*, FITNES-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 in *population*, according to FITNESS-FN

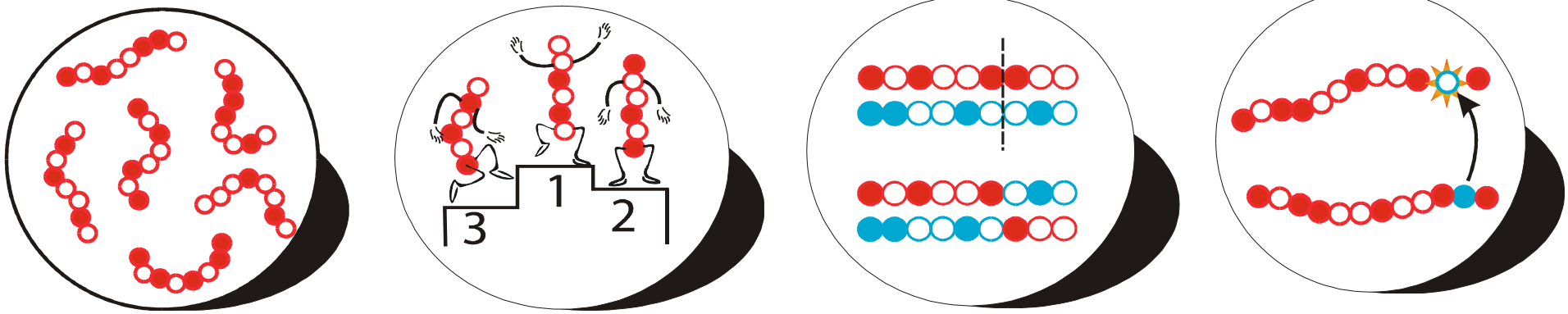
function REPRODUCE(*x*, *y*) returns an individual
  inputs:*x*, *y*, parent individuals
  n ← LENGTH(*x*);
  c ← random number from 1 to *n*
  return APPEND(SUBSTRING(*x*, 1, *c*), SUBSTRING(*y*, *c* + 1, *n*))



Genetic Algorithms (cont.)



- Main issues
  - Encoding schemes
    - Representation of problem states.
  - Size of population
    - Too small → converging too quickly, and vice versa.
  - Fitness function
    - The objective function for optimization/maximization.
    - Ranking members in a population.



A Genetic Algorithm for TSP

Crossover Phase

- Sequence preserving crossover (SPX)

create offspring using the common maximum subsequence of the indices of the two parents.

A=123||5748||69

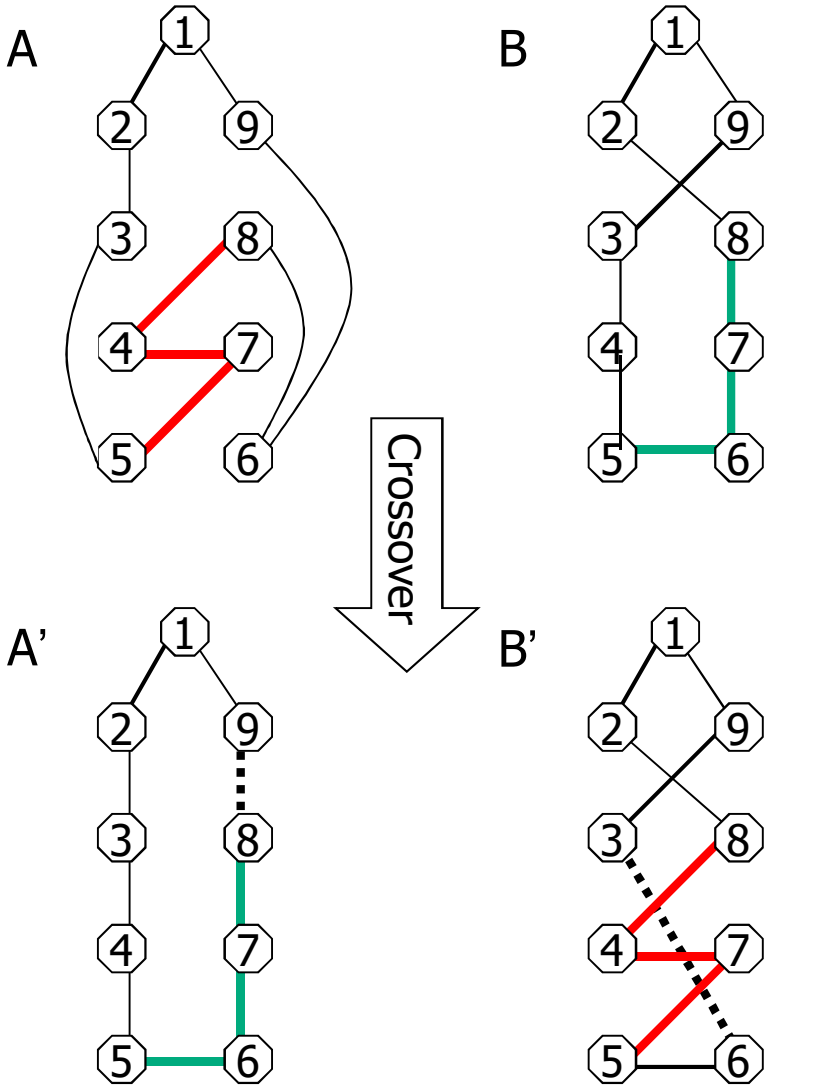
X

B=934||5678||21

X

A'=1234||5678||9

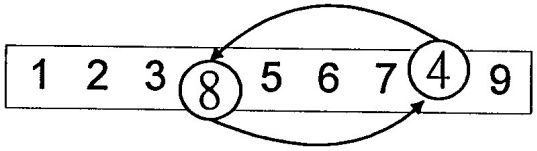
B'=936||5748||21



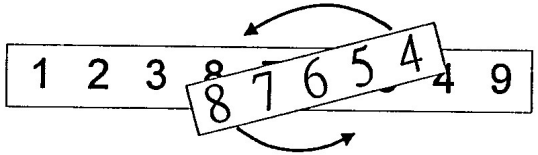
A Genetic Algorithm for TSP

Mutation Phase

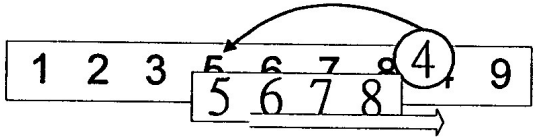
- Point mutation

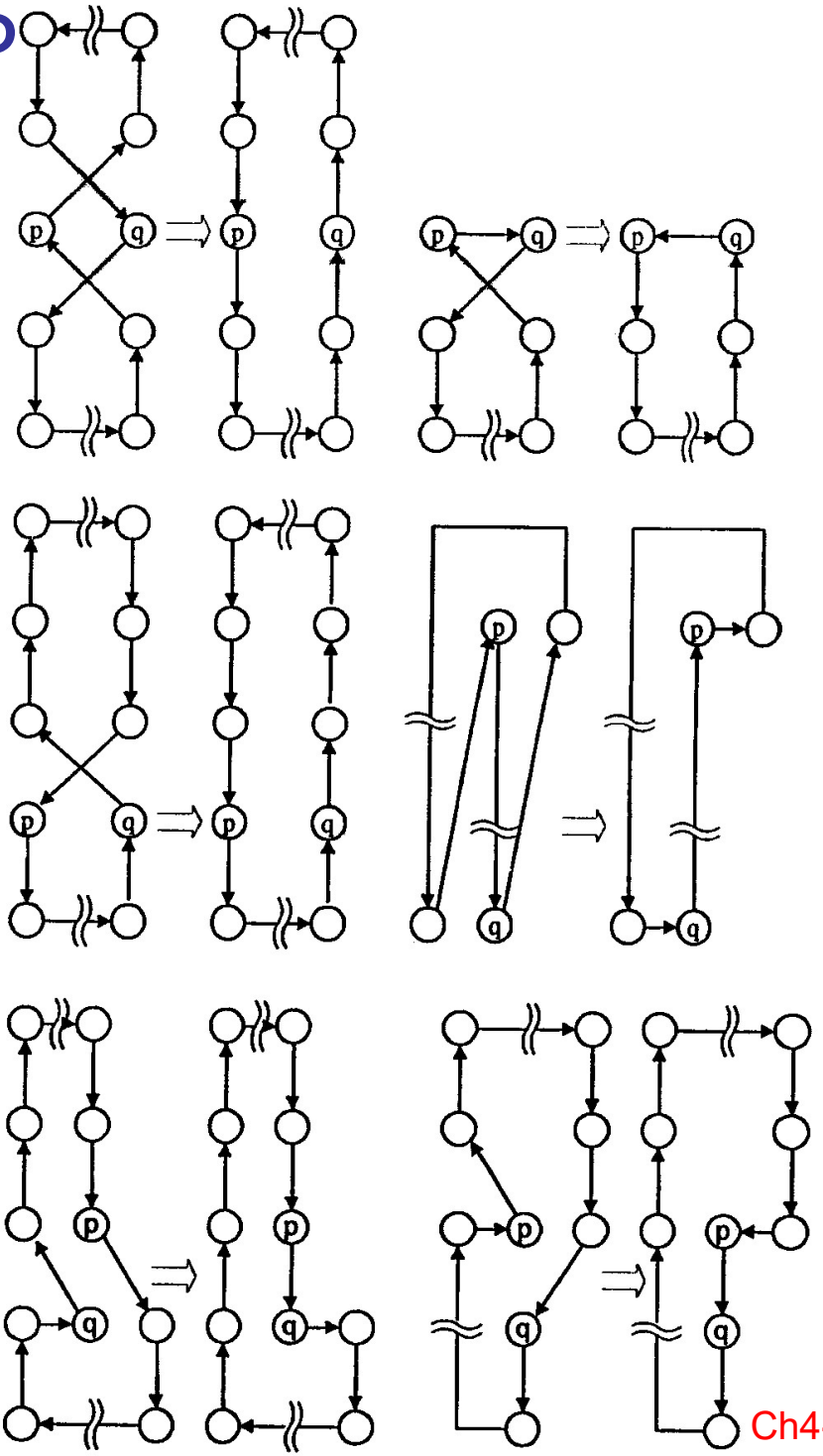


- Inversion mutation



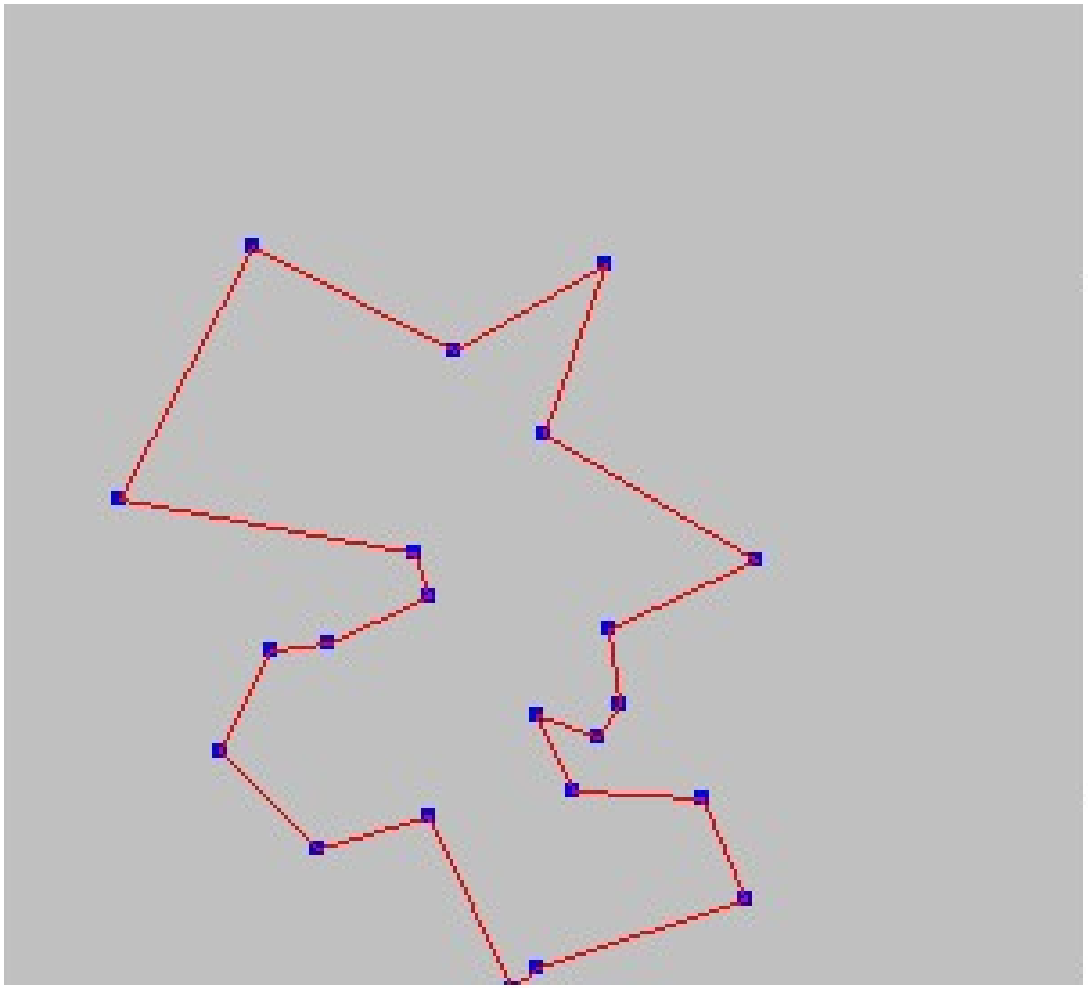
- Shift mutation





A fast TSP solver using a genetic algorithm

http://eecs.wsu.edu/~cook/ai/lectures/applets/gatsp/TSP.html



Start Clear

Generation: 34186

Length of Route: 2.8749

# of Cities: 22

Population: 30

Selection [%]: 30

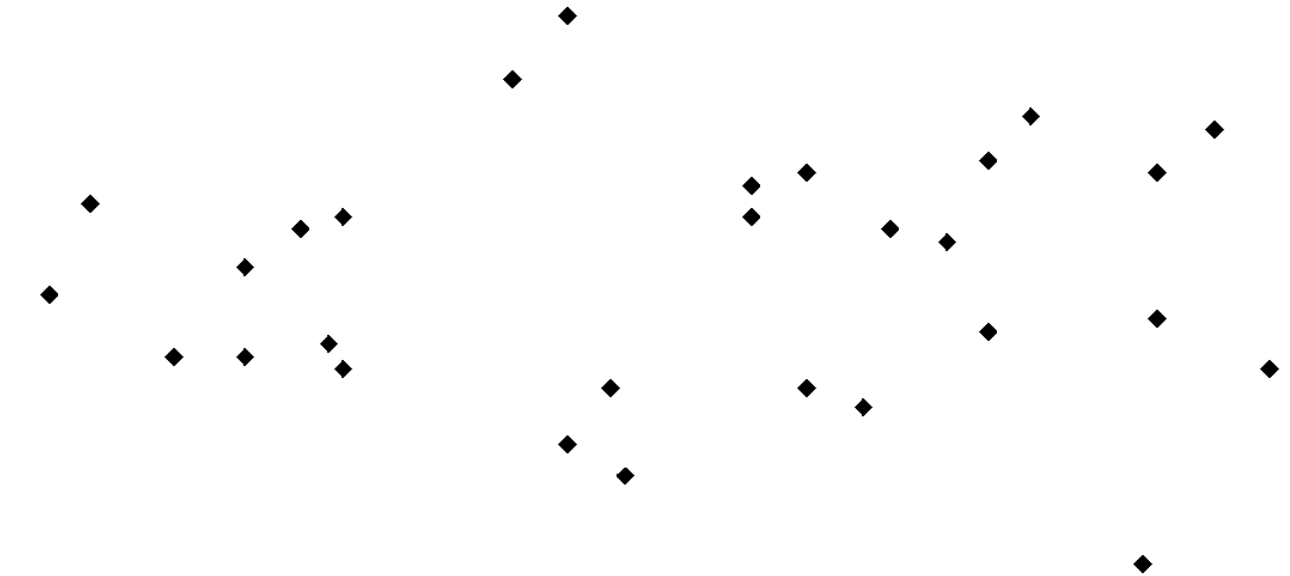
2 opt [%]: 20

Written by Hiroaki Sengoku

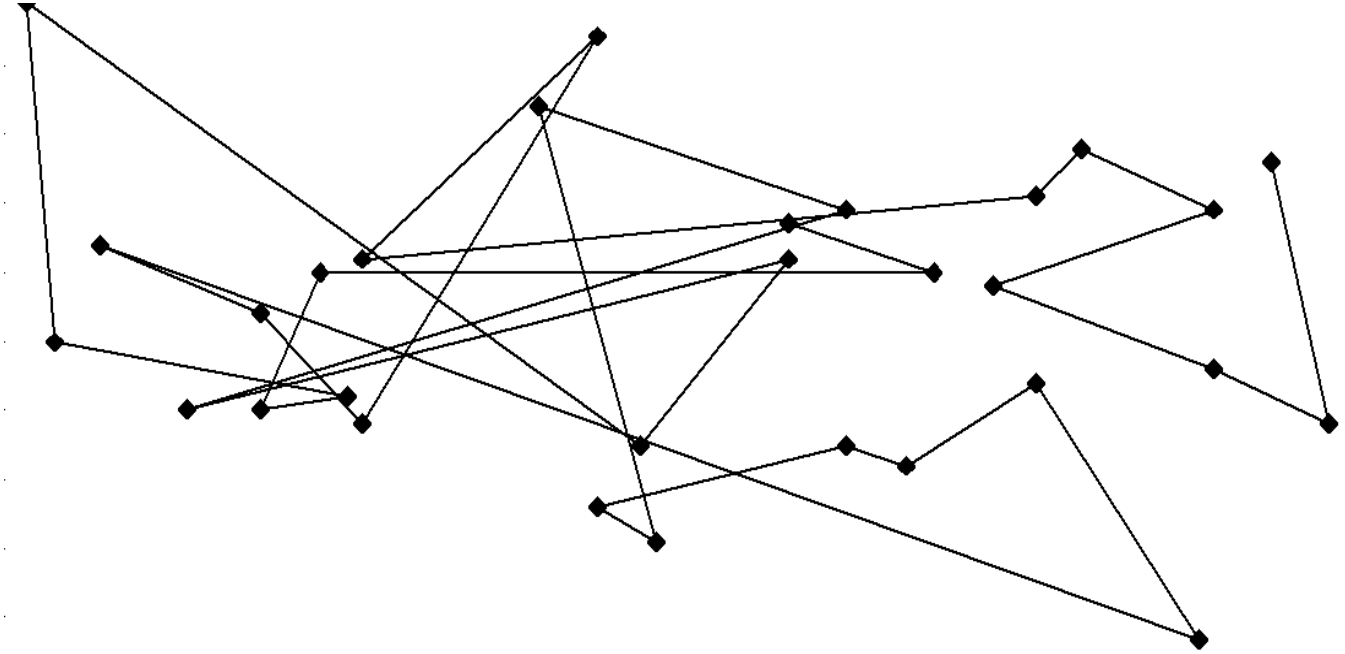
(C)1996-7 by Hitachi, Ltd.

Version 1.3

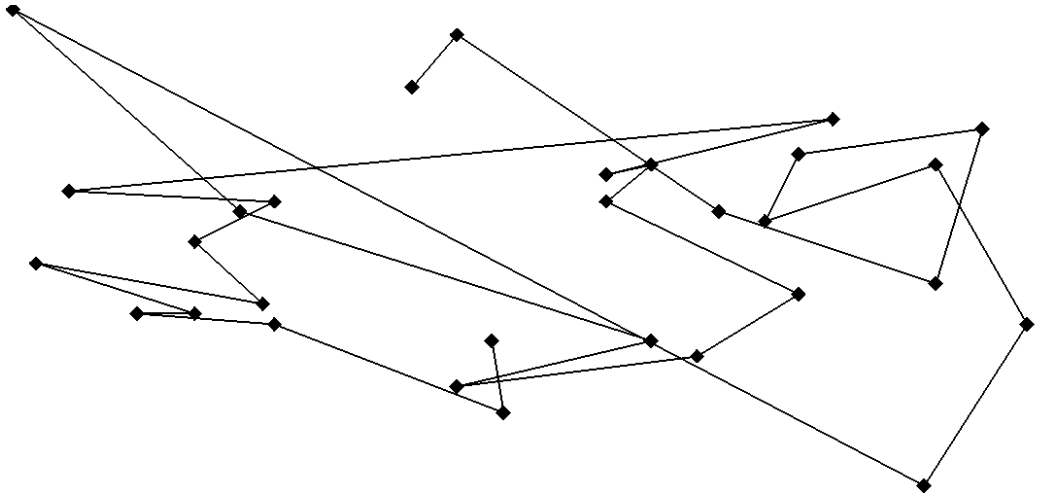
TSP Example: 30 Cities



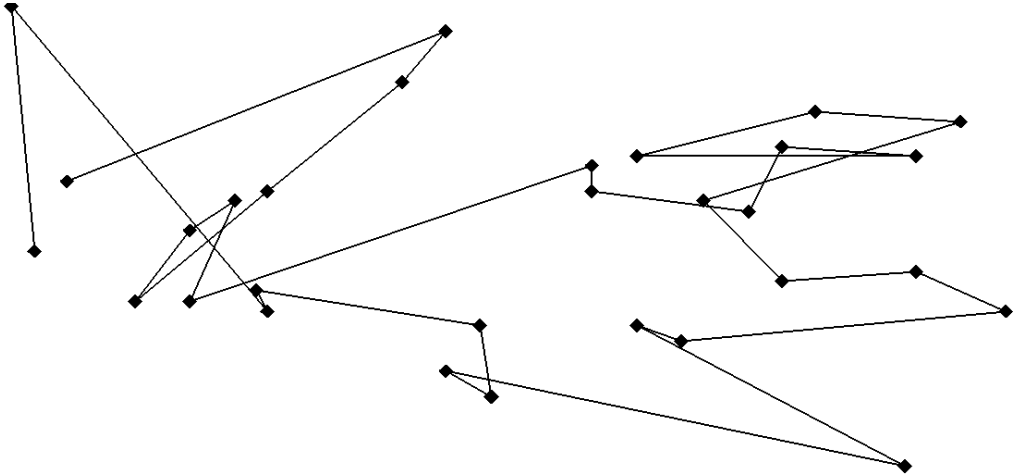
Solution$_i$ (Distance = 941)



Solution$_j$ (Distance = 800)



Solution$_k$ (Distance = 652)



Best Solution (Distance = 420)