MetaHeuristics

INF273

#5: Local Search

Ahmad Hemmati

Optimization Group
Dept. of Informatics
University of Bergen

Spring Semester
2022

Agenda

- Neighborhood

- Local Search Operators
  - ✓ Flip
  - ✓ Swap (2-exchange)
  - ✓ 2-Opt, 3-Opt, … k-opt
  - ✓ …

- Local Search and Hill Climbing

Local Search

- Given a Solution: How to Find a Better One?
- Modification of a given solution gives a "neighbor solution"
- A certain set of operations on a solution gives a set of neighbor solutions, a neighborhood
- Evaluations of neighbors
  - Objective function value
  - Feasibility?

Local Search Operators

- Also known as Neighborhood Operator
- Neighborhoods are most often defined by a given operation on a solution: $N(s)$
- Often simple operations
  - Remove an element
  - Add an element
  - Interchange two or more elements of a solution

Local Search Operators

- Example: Knapsack Problem (10 items, capacity: 100)
- Given solution 0010100000, is feasible. Value = 73

	1	2	3	4	5	6	7	8	9	10
Value	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52
Given Solution	0	0	1	0	1	0	0	0	0	0

Local Search Operators

- Natural operator: "*Flip*" a bit, i.e.
  - If the item is in the knapsack, take it out
  - If the item is not in the knapsack, include it

	1	2	3	4	5	6	7	8	9	10
Value	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52
Given Solution	0	0	1	0	1	0	0	0	0	0

- Given solution: 0010100000 , value = 73, size = 70

Local Search Operators

- Natural operator: "*Flip*" a bit, i.e.
  - If the item is in the knapsack, take it out
  - If the item is not in the knapsack, include it

	1	2	3	4	5	6	7	8	9	10
Value	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52
Given Solution	0	0	1	0	1	0	0	0	0	0

- Given solution: 0010100000 , value = 73, size = 70

Local Search Operators

- Natural operator: "*Flip*" a bit, i.e.
  - If the item is in the knapsack, take it out
  - If the item is not in the knapsack, include it

	1	2	3	4	5	6	7	8	9	10
Value	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52
Given Solution	0	0	1	0	1	0	0	0	0	0
Neighbor Solution	0	1	1	0	1	0	0	0	0	0

- Given solution: 0010100000 , value = 73, size = 70
- Neighbor solution: 0110100000 , value = 105, size = 96

Local Search Operators

	1	2	3	4	5	6	7	8	9	10
Value	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52
Given Solution	0	0	1	0	1	0	0	0	0	0
Neighbor Solution	0	1	1	0	1	0	0	0	0	0

- Some Neighbors:
  - 0110100000 , value = 105, size = 96

Local Search Operators

	1	2	3	4	5	6	7	8	9	10
Value	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52
Given Solution	0	0	1	0	1	0	0	0	0	0
Neighbor Solution	0	1	1	0	1	0	0	0	0	0
Another Neighbor	1	0	1	0	1	0	0	0	0	0

- Some Neighbors:
  - 0110100000 , value = 105, size = 96

Local Search Operators

	1	2	3	4	5	6	7	8	9	10
Value	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52
Given Solution	0	0	1	0	1	0	0	0	0	0
Neighbor Solution	0	1	1	0	1	0	0	0	0	0
Another Neighbor	1	0	1	0	1	0	0	0	0	0

- Some Neighbors:
  - 0110100000 , value = 105, size = 96
  - 1010100000 , value = 152, not feasible

	1	2	3	4	5	6	7	8	9	10
Value	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52
Given Solution	0	0	1	0	1	0	0	0	0	0
Neighbor Solution	0	1	1	0	1	0	0	0	0	0
Neighbor Solution	1	0	1	0	1	0	0	0	0	0

- Some Neighbors:
  - 0110100000 , value = 105, size = 96
  - 1010100000 , value = 152, not feasible

Local Search Operators

	1	2	3	4	5	6	7	8	9	10
Value	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52
Given Solution	0	0	1	0	1	0	0	0	0	0
Neighbor Solution	0	1	1	0	1	0	0	0	0	0
Neighbor Solution	1	0	1	0	1	0	0	0	0	0
Neighbor Solution	0	0	1	0	0	0	0	0	0	0

- Some Neighbors:
  - 0110100000 , value = 105, size = 96
  - 1010100000 , value = 152, not feasible
  - 0010000000 , value = 47, size = 48

Local Search Operators

	1	2	3	4	5	6	7	8	9	10
Value	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52
Given Solution	0	0	1	0	1	0	0	0	0	0
Neighbor Solution	0	1	1	0	1	0	0	0	0	0
Neighbor Solution	1	0	1	0	1	0	0	0	0	0
Neighbor Solution	0	0	1	0	0	0	0	0	0	0

- Some Neighbors:
  - 0110100000 , value = 105, size = 96
  - 1010100000 , value = 152, not feasible
  - 0010000000 , value = 47, size = 48
- How many neighbors?

Local Search Operators

	1	2	3	4	5	6	7	8	9	10
Value	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52
Given Solution	0	0	1	0	1	0	0	0	0	0
Neighbor Solution	0	1	1	0	1	0	0	0	0	0
Neighbor Solution	1	0	1	0	1	0	0	0	0	0
Current Solution	0	0	1	0	0	0	0	0	0	0

- 0010100000 , value = 73, size = 70
- 0110100000 , value = 105, size = 96
- 1010100000 , value = 152, not feasible
- 0010000000 , value = 47, size = 48

Local Search Operators

	1	2	3	4	5	6	7	8	9	10
Value	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52
Given Solution	0	0	1	0	1	0	0	0	0	0
Current Solution	0	0	1	0	0	0	0	0	0	0

– 0010100000 , value = 73, size = 70

– 0010000000 , value = 47, size = 48

Local Search Operators

	1	2	3	4	5	6	7	8	9	10
Value	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52
Given Solution	0	0	1	0	1	0	0	0	0	0
Current Solution	0	0	1	0	0	0	0	0	0	0
Current Solution	0	0	1	0	0	0	0	0	0	1

- 0010100000 , value = 73, size = 70
- 0010000000 , value = 47, size = 48
- 0010000001 , value = 106, size = 100

Local Search Operators

	1	2	3	4	5	6	7	8	9	10
Value	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52
Given Solution	0	0	1	0	1	0	0	0	0	0
Current Solution	0	0	1	0	0	0	0	0	0	0
Current Solution	0	0	1	0	0	0	0	0	0	1

- 0010100000 , value = 73, size = 70
- 0010000000 , value = 47, size = 48
- 0010000001 , value = 106, size = 100

- The global best?

Local Search Operators

	1	2	3	4	5	6	7	8	9	10
Value	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52
Given Solution	0	0	1	0	1	0	0	0	0	0
Current Solution	0	0	1	0	0	0	0	0	0	0
Current Solution	0	0	1	0	0	0	0	0	0	1

Best Solution	0	1	0	0	1	0	0	0	0	1

- 0010100000 , value = 73, size = 70
- 0010000000 , value = 47, size = 48
- 0010000001 , value = 106, size = 100

- 0100100001 , value = 117, size = 100

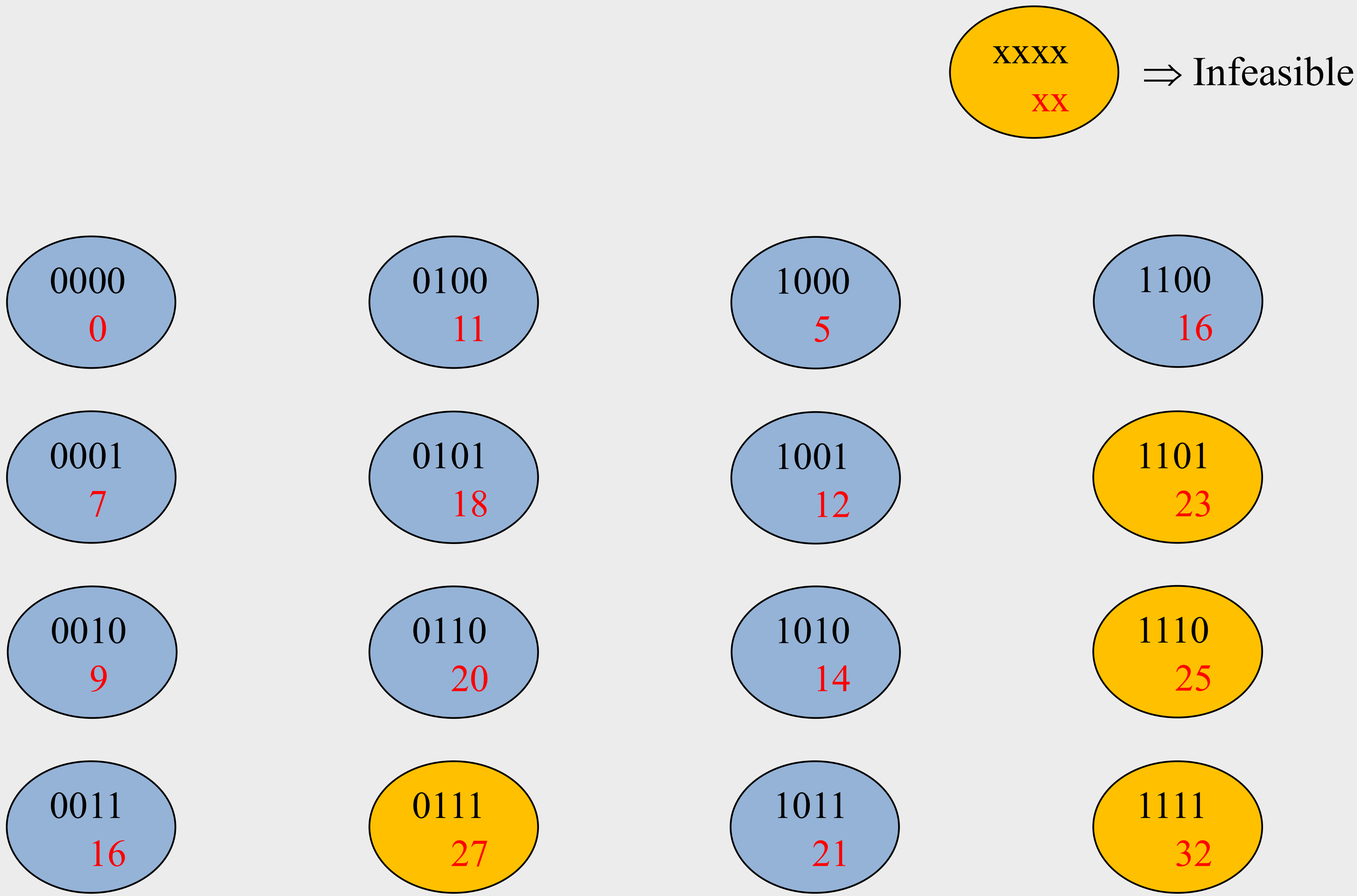
SEARCH SPACE

- The search space is the set of solutions (2^N)

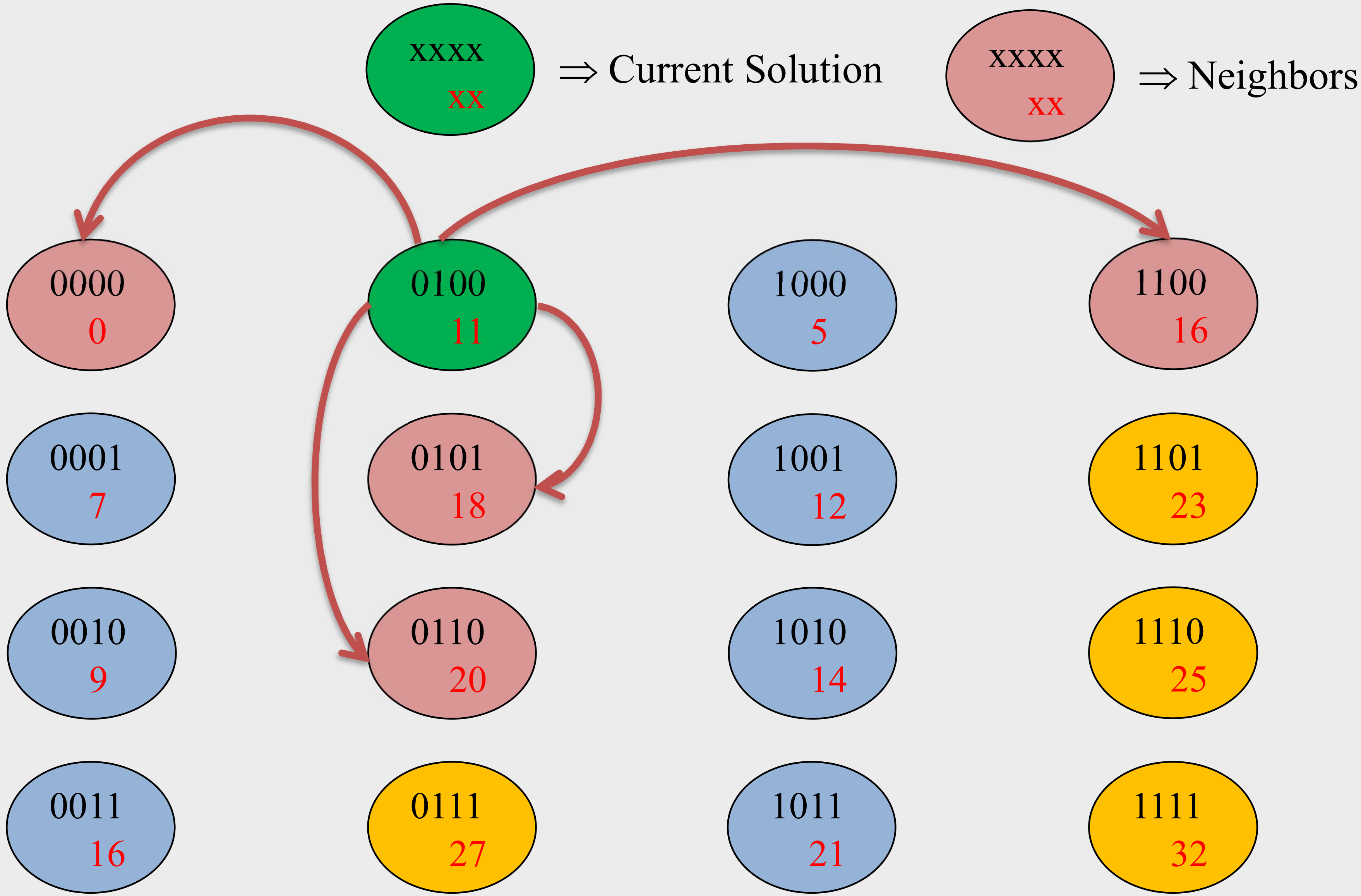
xxxx ⇒ Solution
Obj. Fun. Value

0000 0	0100 11	1000 5	1100 16
0001 7	0101 18	1001 12	1101 23
0010 9	0110 20	1010 14	1110 25
0011 16	0111 27	1011 21	1111 32

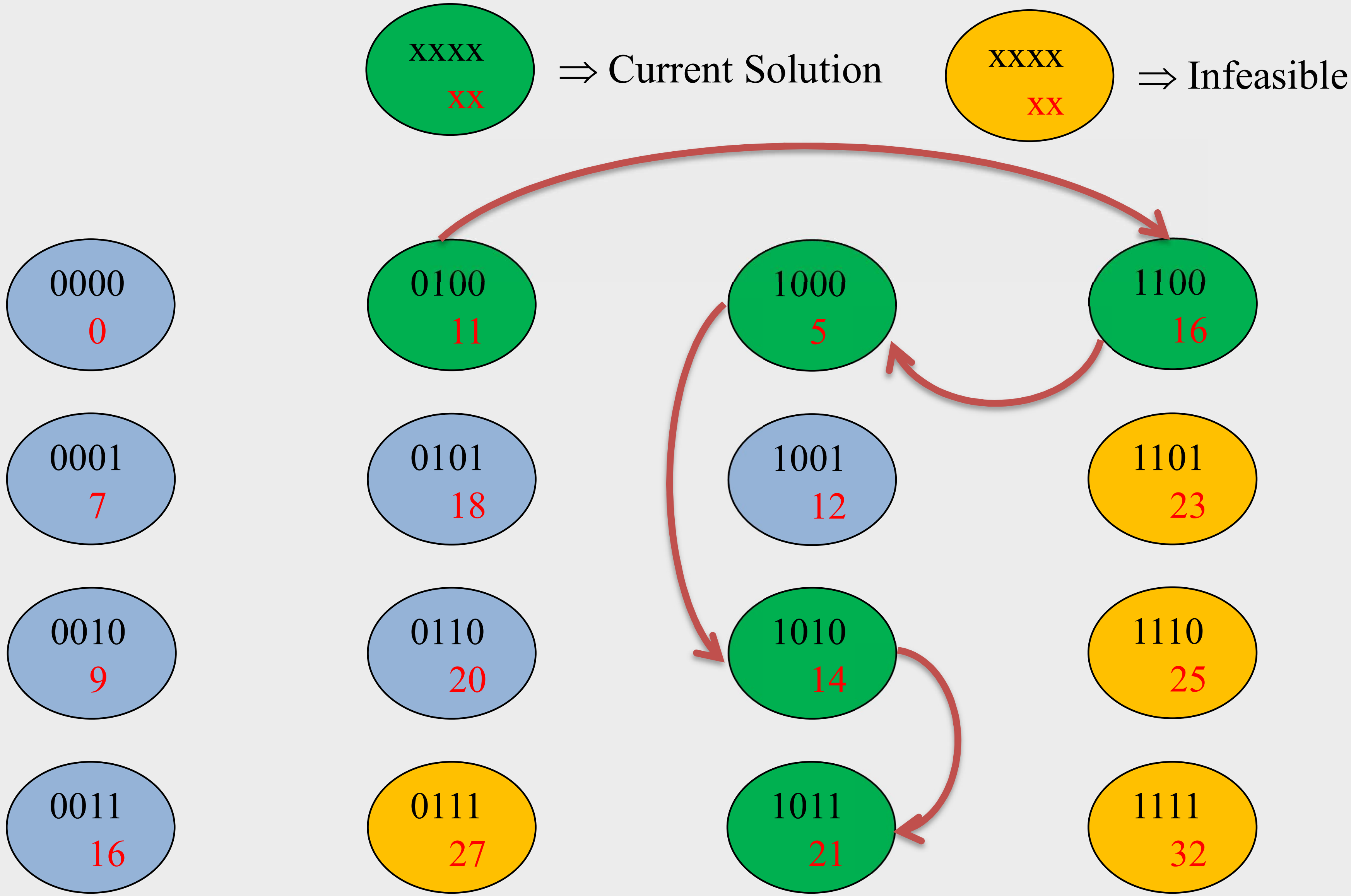
Feasible/Infeasible Space



Neighbors



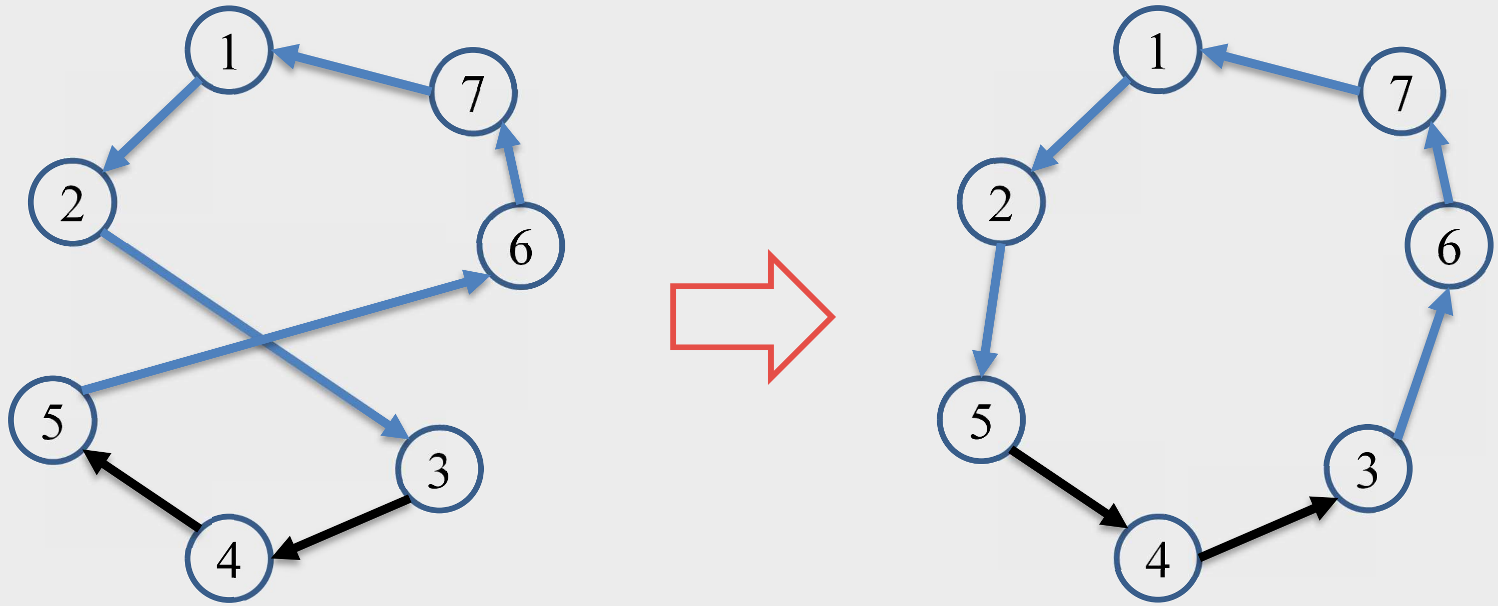
xxxx
xx
⇒ Current Solution
xxxx
xx
⇒ Neighbors
0000
0
0100
11
1000
5
1100
16
0001
7
0101
18
1001
12
1101
23
0010
9
0110
20
1010
14
1110
25
0011
16
0111
27
1011
21
1111
32



xxxx
xx
⇒ Current Solution
xxxx
xx
⇒ Infeasible
0000
0
0100
11
1000
5
1100
16
0001
7
0101
18
1001
12
1101
23
0010
9
0110
20
1010
14
1110
25
0011
16
0111
27
1011
21
1111
32

- Swap (2-exchange)

  (Example: TSP)

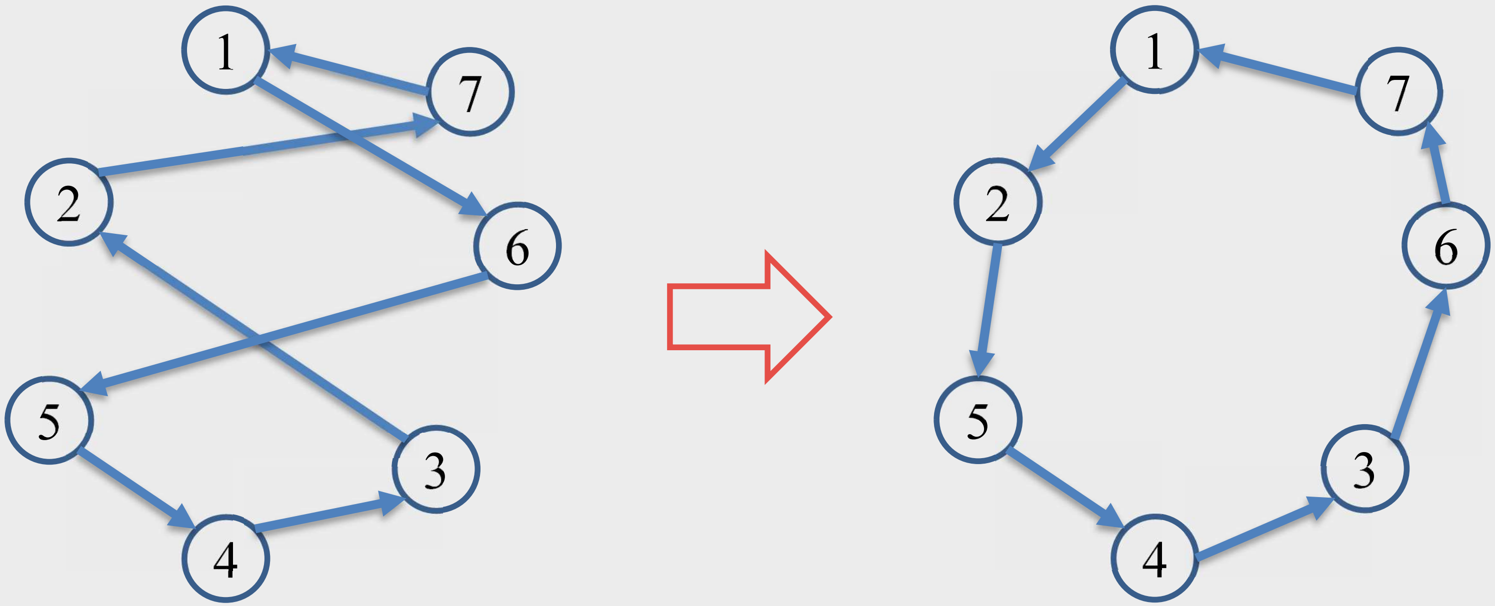


1	2	3	4	5	6	7

1	2	5	4	3	6	7

- Swap (2-exchange)

(Example: TSP)

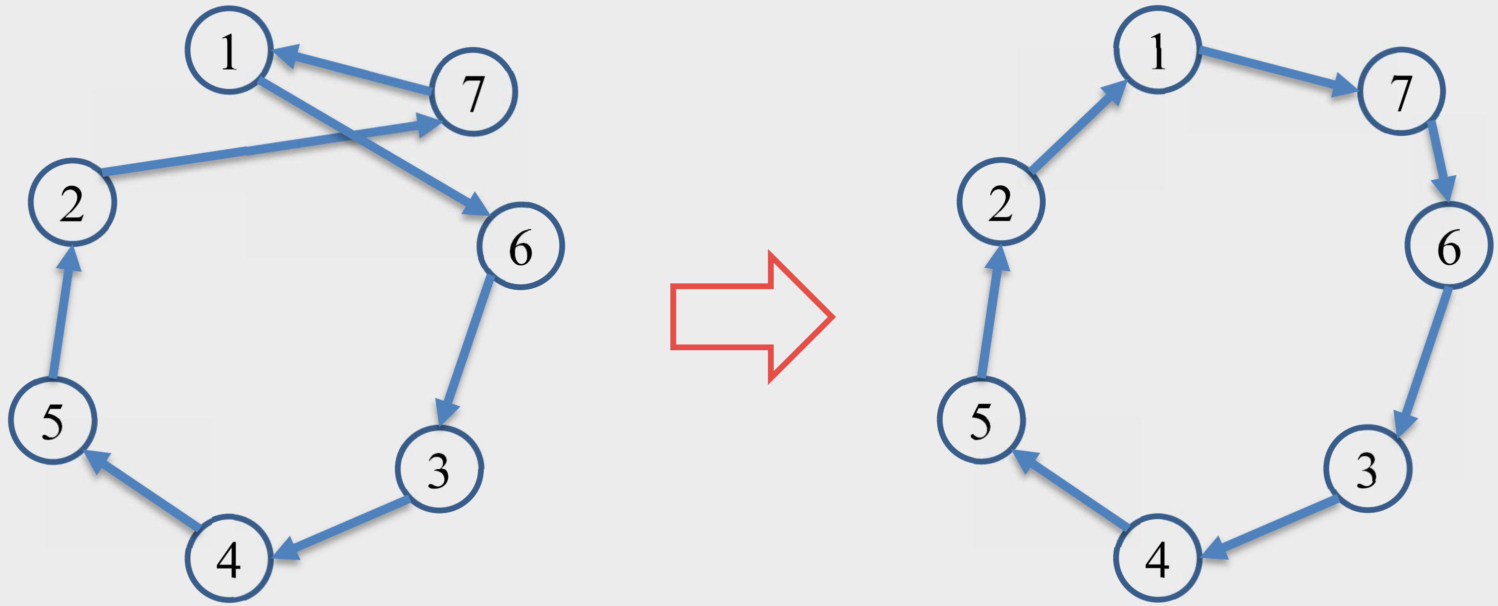


1	6	5	4	3	2	7

1	2	5	4	3	6	7

- Swap (2-exchange)

(Example: TSP)

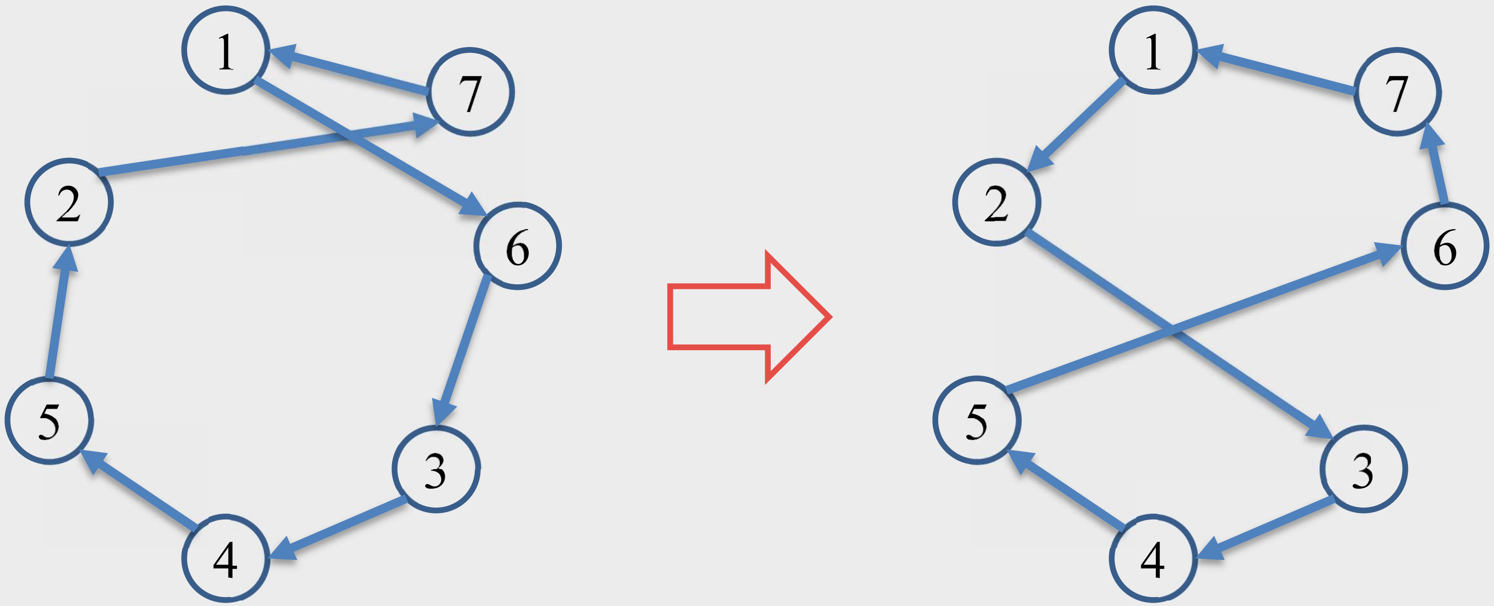


1	6	3	4	5	2	7

7	6	3	4	5	2	1

- Swap (2-exchange)

  (Example: TSP)



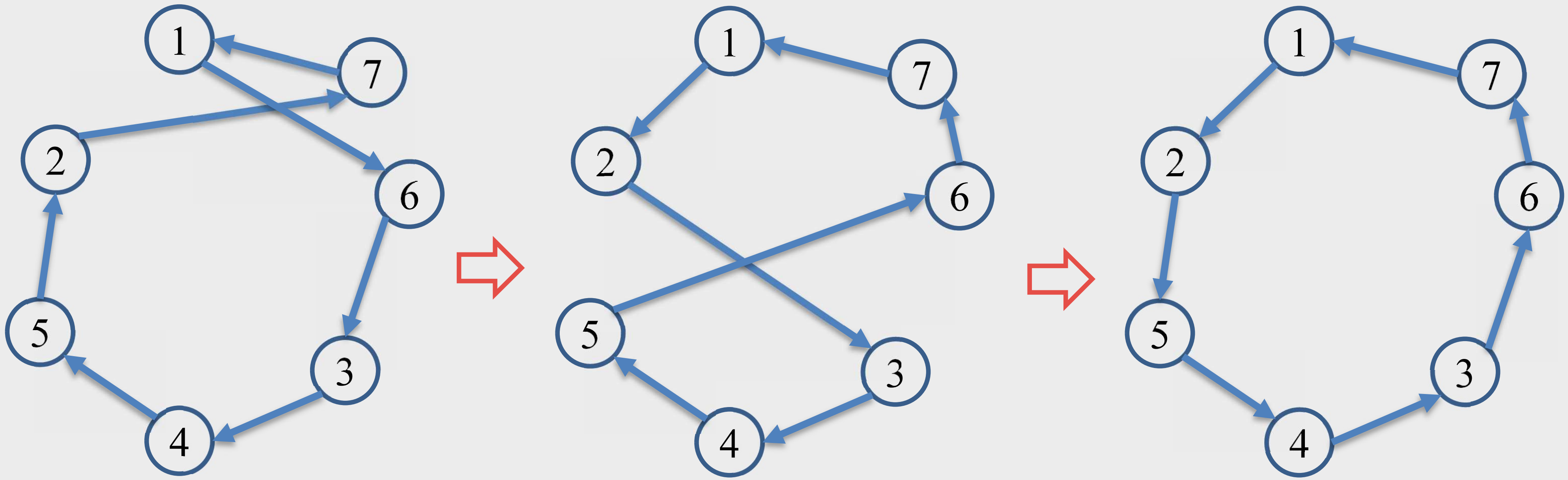
1	6	3	4	5	2	7

1	2	3	4	5	6	7

Local Search Operators

- Swap (2-exchange)

  (Example: TSP)



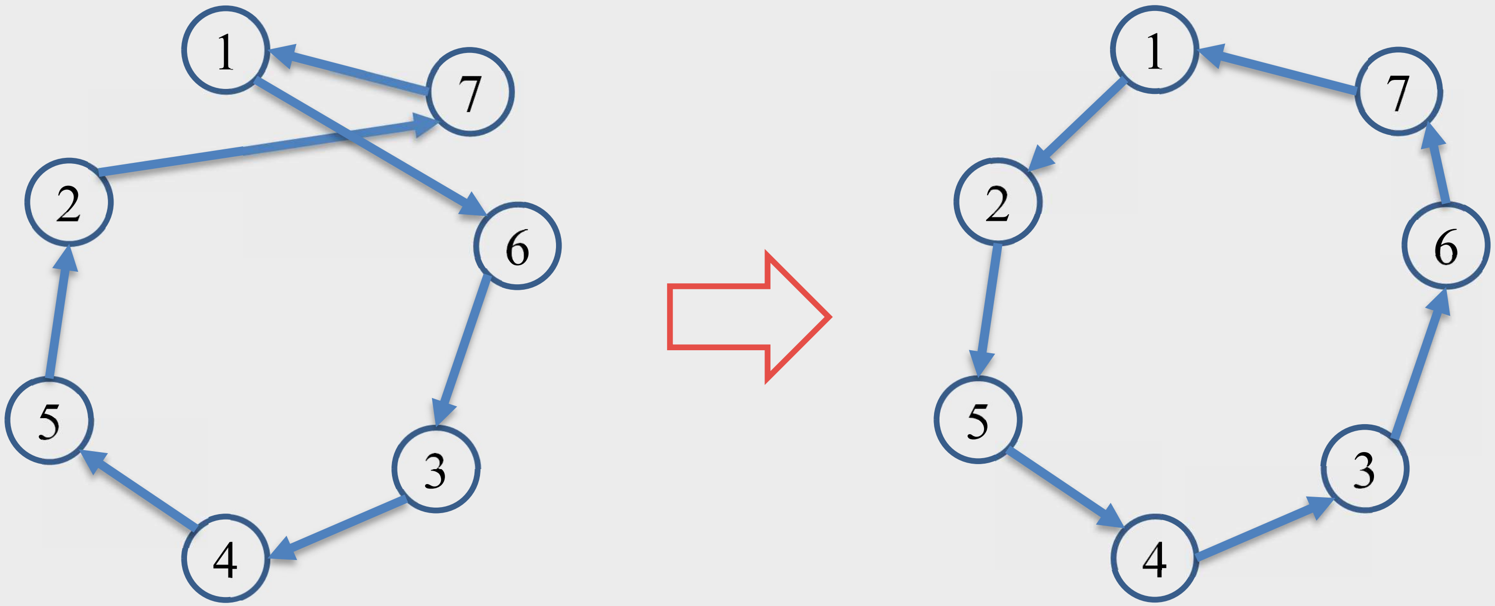
1	6	3	4	5	2	7

1	2	3	4	5	6	7

1	2	5	4	3	6	7

- Swap (2-exchange) with reversing

(Example: TSP)

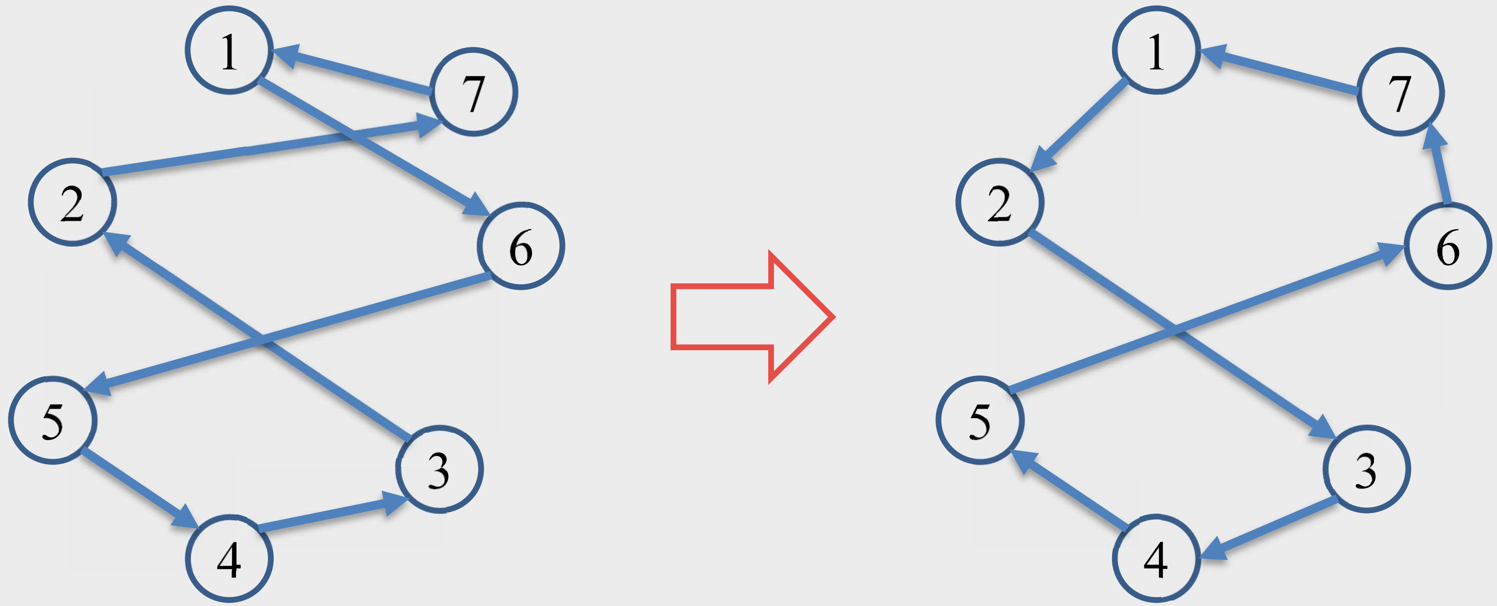


1	6	3	4	5	2	7

1	2	5	4	3	6	7

- Swap (2-exchange) with reversing

(Example: TSP)

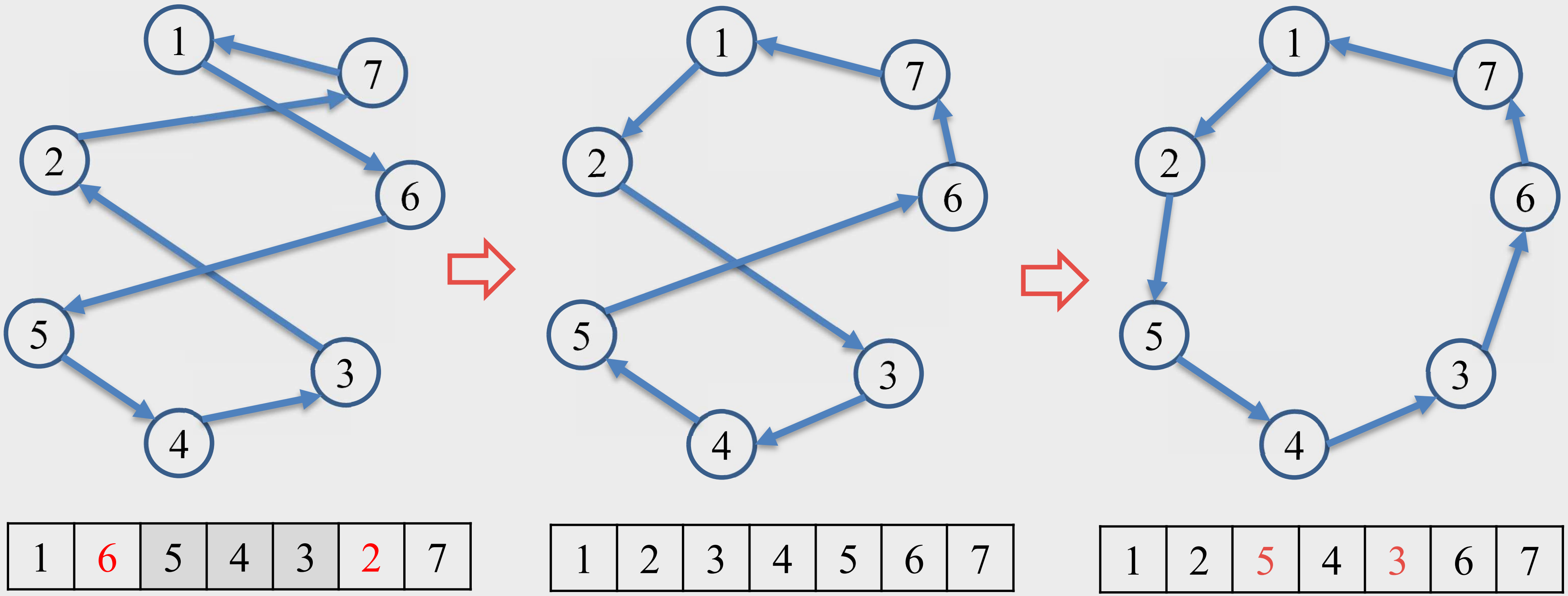


1	6	5	4	3	2	7

1	2	3	4	5	6	7

- Swap (2-exchange) with reversing

  (Example: TSP)



Local Search Operators

- *2-opt*

  (Example: TSP)

  – Note: a finite sequence of “swap (2-exchange) with or without reversing" can generate any tour (for TSP), including the optimum tour

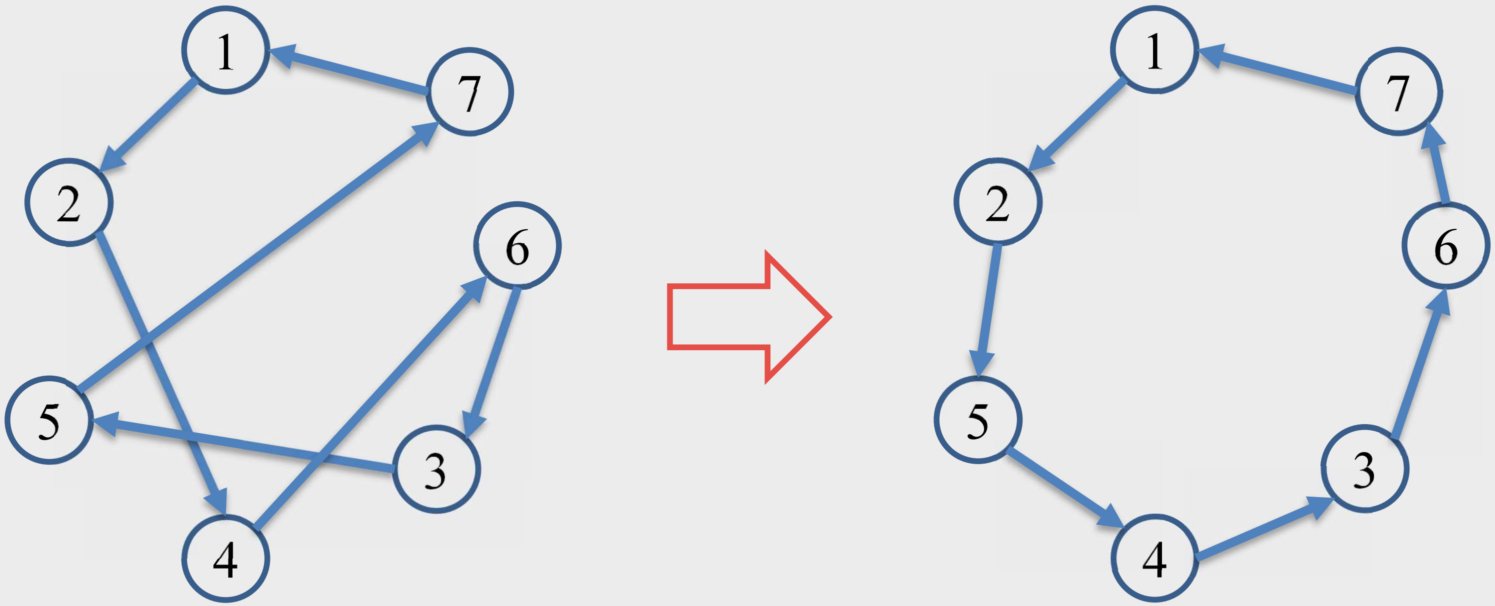
  – Strategy:

    - Select the best swap among N*(N-1)/2 neighbors (2-move neighborhood)
    - Repeat this process until no improvement can be made

Local Search Operators

- *3-exchange → 3-opt*

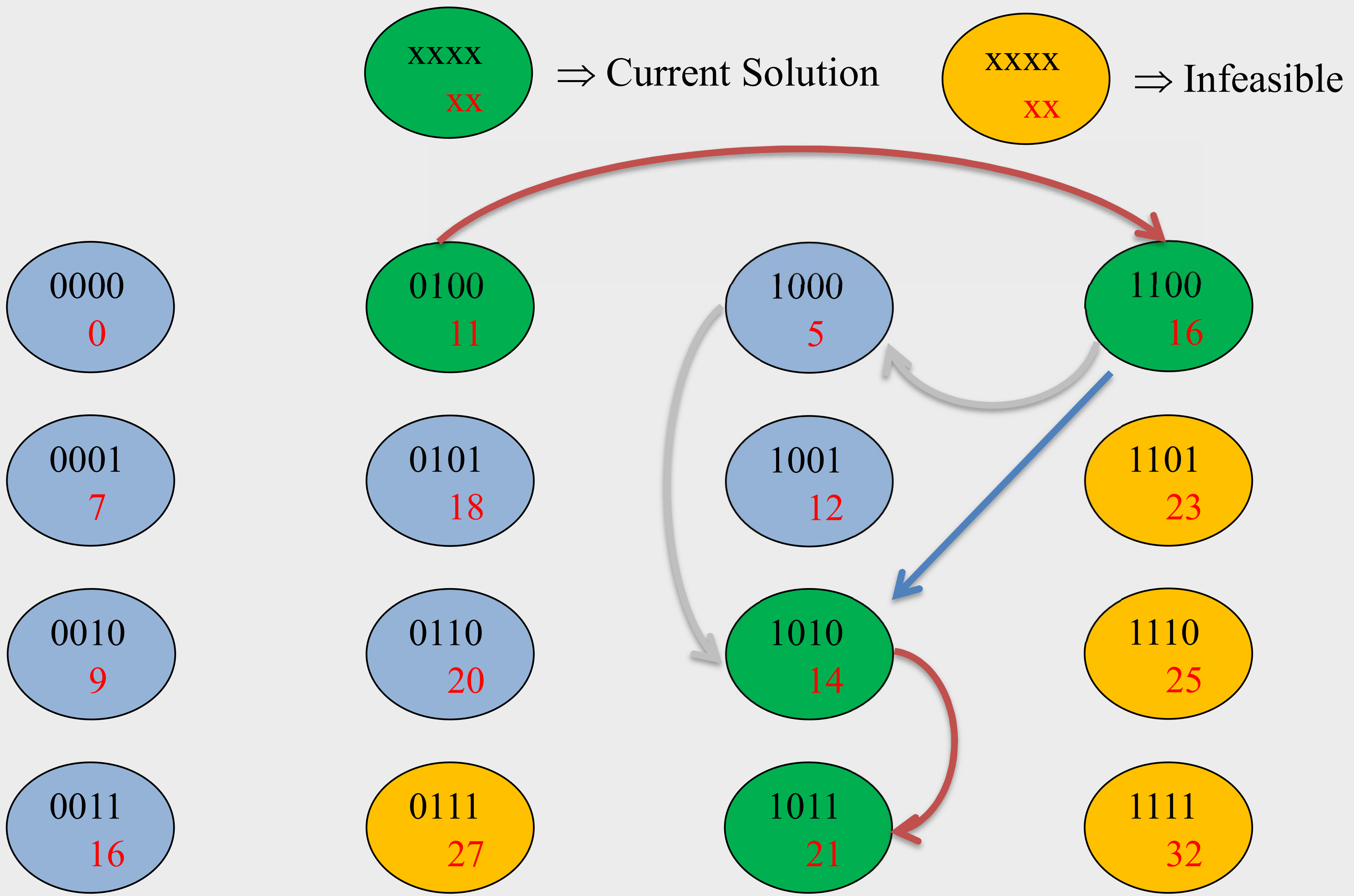
  (Example: TSP)



1	2	4	6	3	5	7

1	2	5	4	3	6	7

Local Search Operators



Local Search–Advantages

- For many problems, it is quite easy to design a local search (LS can be applied to almost any problem)
- The idea of improving a solution by making small changes is easy to understand
- The use of neigborhoods sometimes makes the optimal solution seem "close", e.g.:
  - A knapsack has n items
  - The search space has 2^n members
  - From any solution, no more than n flips are required to reach an optimal solution!

Local Search

Local search 1: Best Accept

```
Input: initial solution (s_0)
Input: neighborhood operator N, N(s) ⟶ all neighbors for s
Input: evaluation function f, f(s) ⟶ the cost of s
Current ⟸ s_0
done ⟸ false
while done = false do
    Best_neighbor ⟸ Current
    for each s ∈ N(Current) do
        if f(s) < f(Best_neighbor) then
            Best_neighbor ⟸ s
        end if
    end for
    if Current = Best_neighbor then
        done ⟸ true
    else
        Current ⟸ Best_neighbor
    end if
end while
```

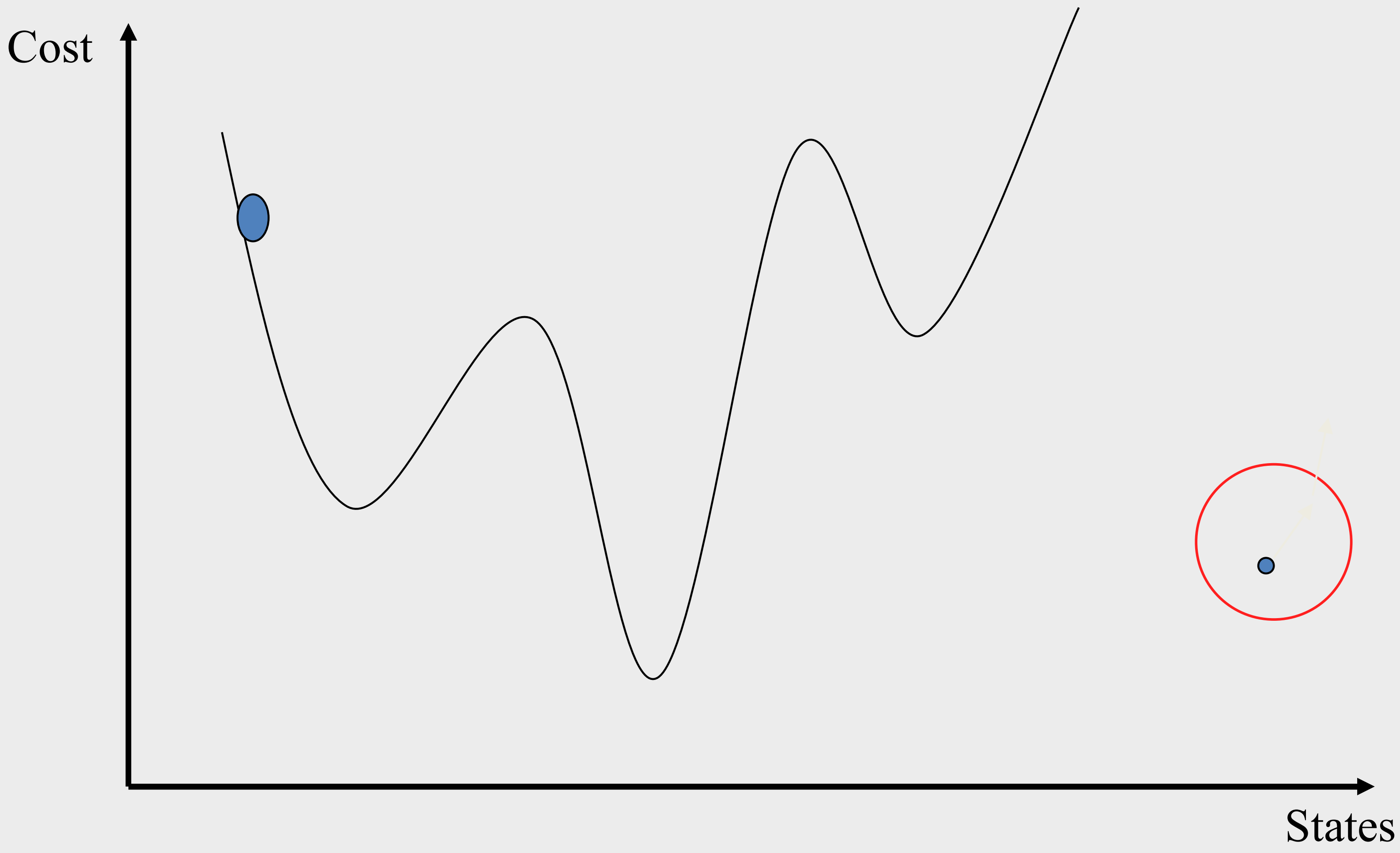
Local search 2: First Accept

```
Input: initial solution (s_0)
Input: neighborhood operator N, N(s) ⟶ all neighbors for s
Input: evaluation function f, f(s) ⟶ the cost of s
Current ⟸ s_0
done ⟸ false
while done = false do
    Best_neighbor ⟸ Current
    for each s ∈ N(Current) do
        if f(s) < f(Best_neighbor) then
            Best_neighbor ⟸ s
            exit the for-loop
        end if
    end for
    if Current = Best_neighbor then
        done ⟸ true
    else
        Current ⟸ Best_neighbor
    end if
end while
```

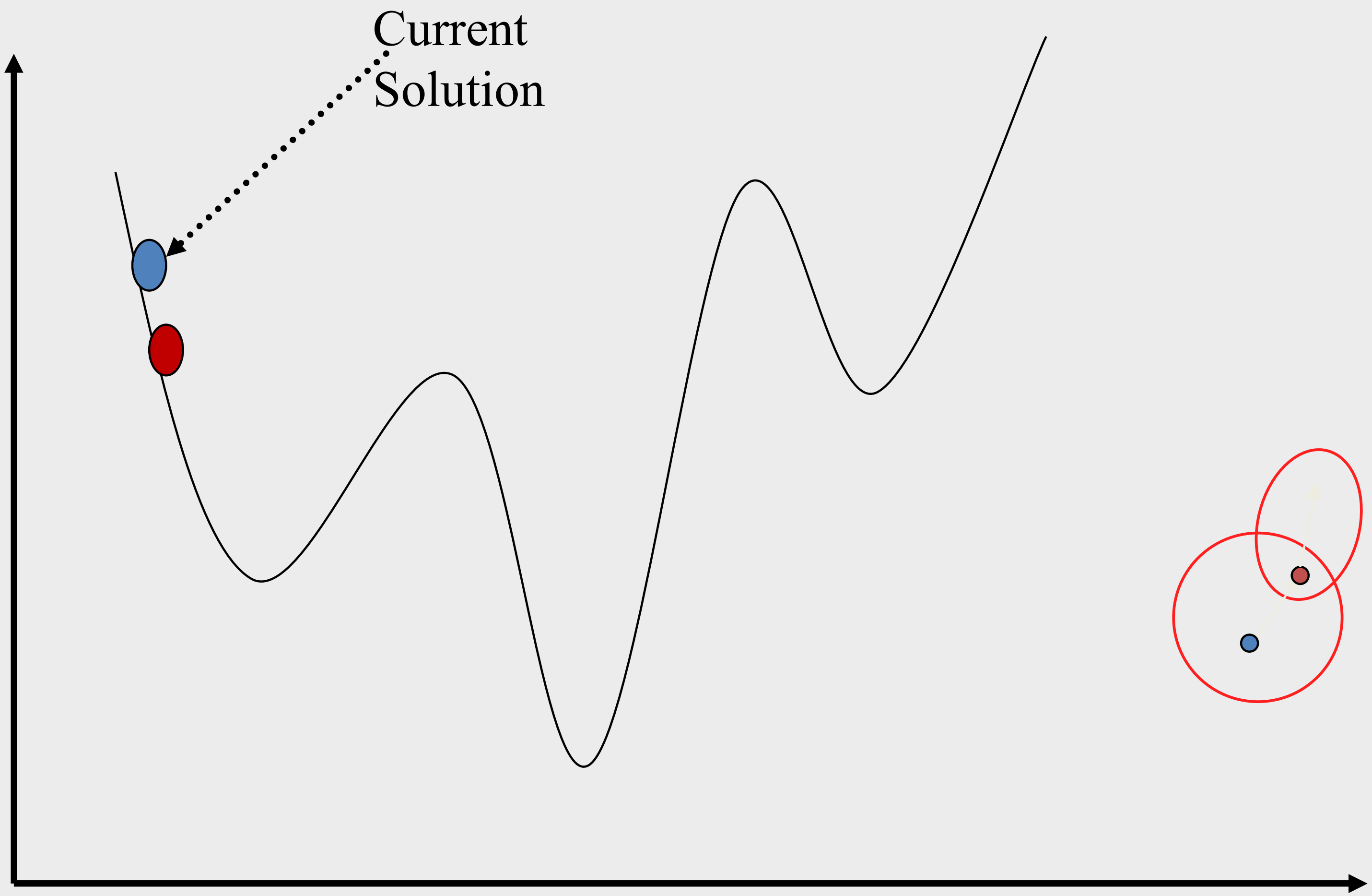
Local Search

- Start with an *initial solution*
- Iteratively search in the neighborhood for better solutions
- Sequence of solutions $s_{k+1} \in N(s_k)$
- Strategy for which solution in the neighborhood that will be accepted as the next solution
- Stopping Criteria
- What happens when the neighborhood does not contain a better solution?

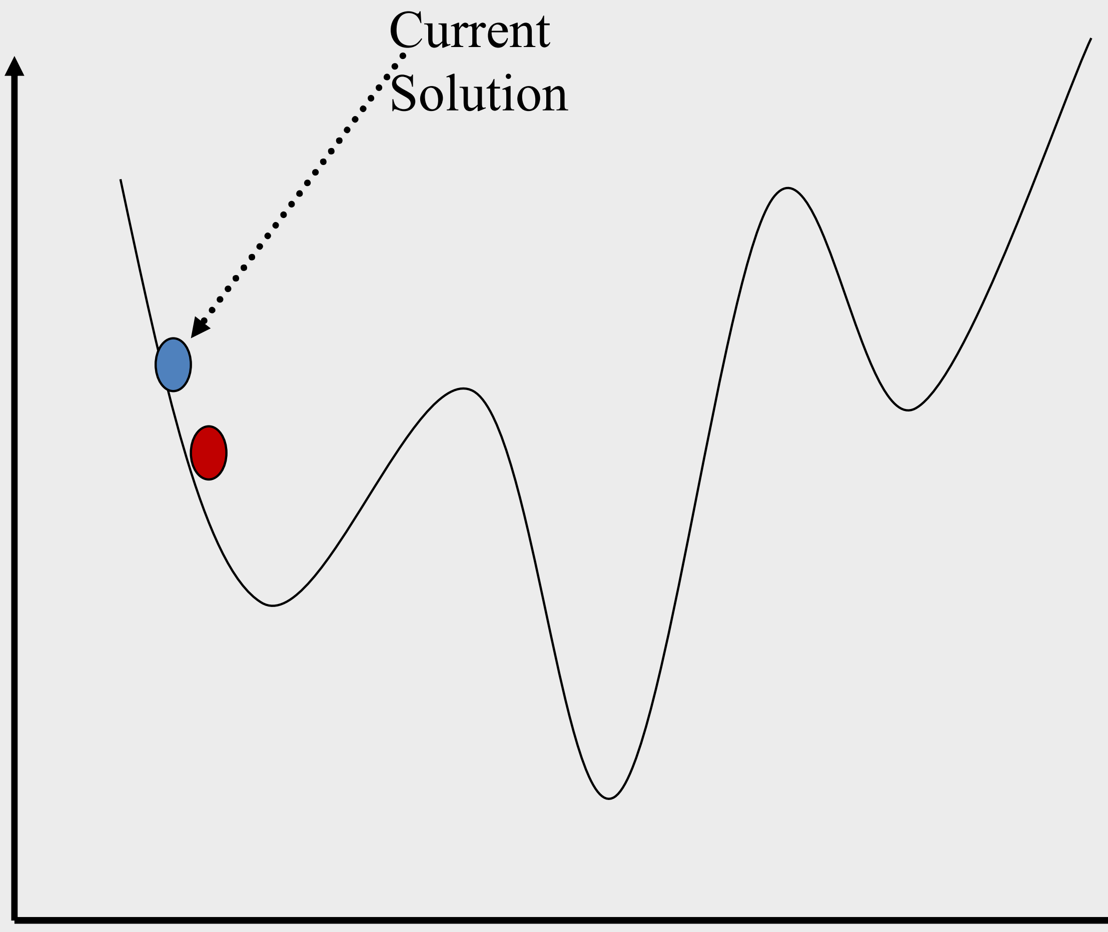
Local Search

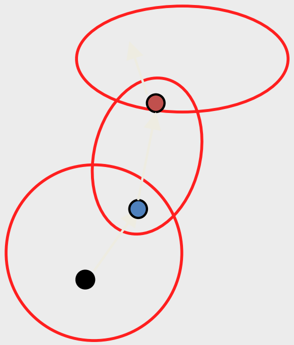


Local Search

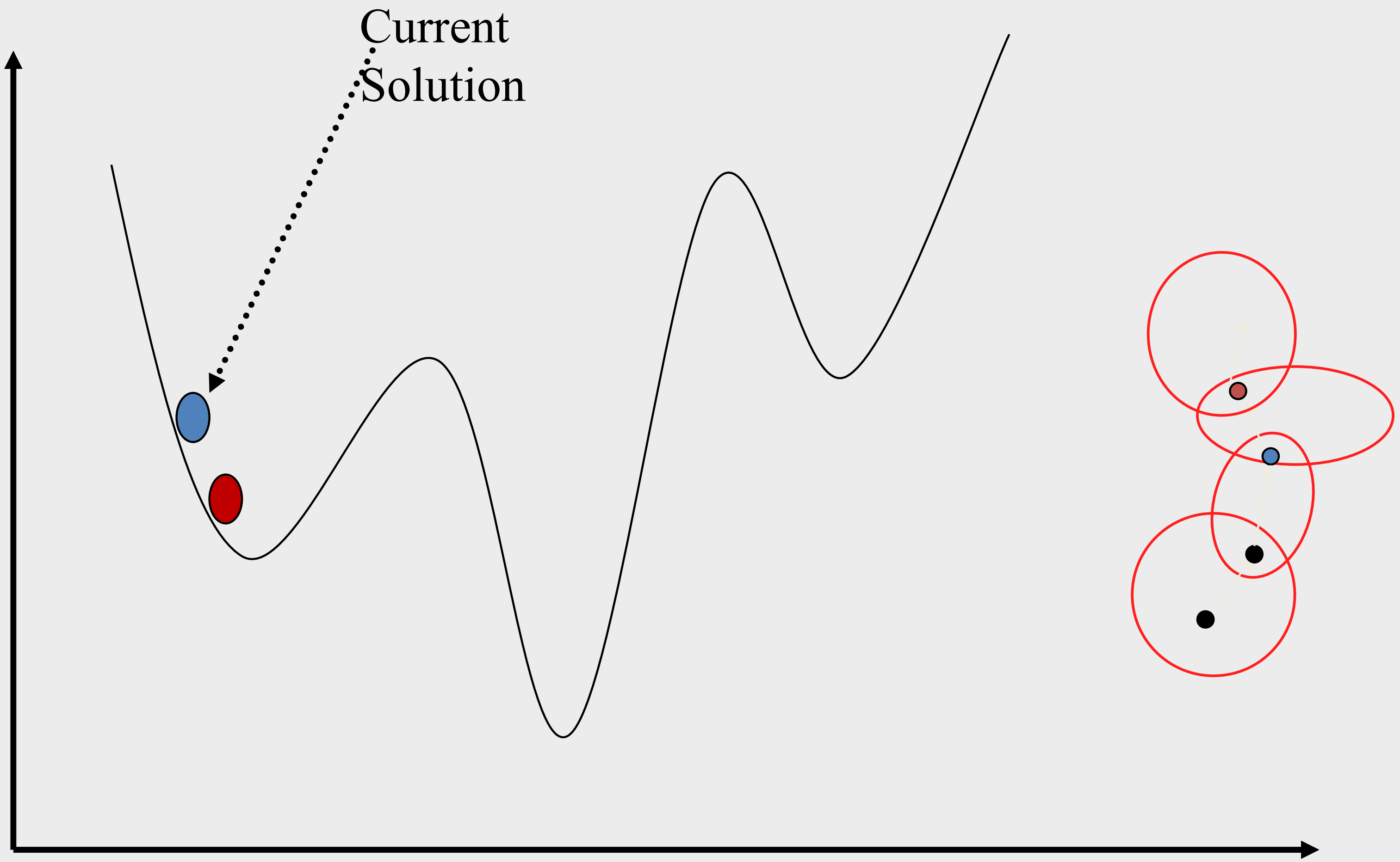


Local Search

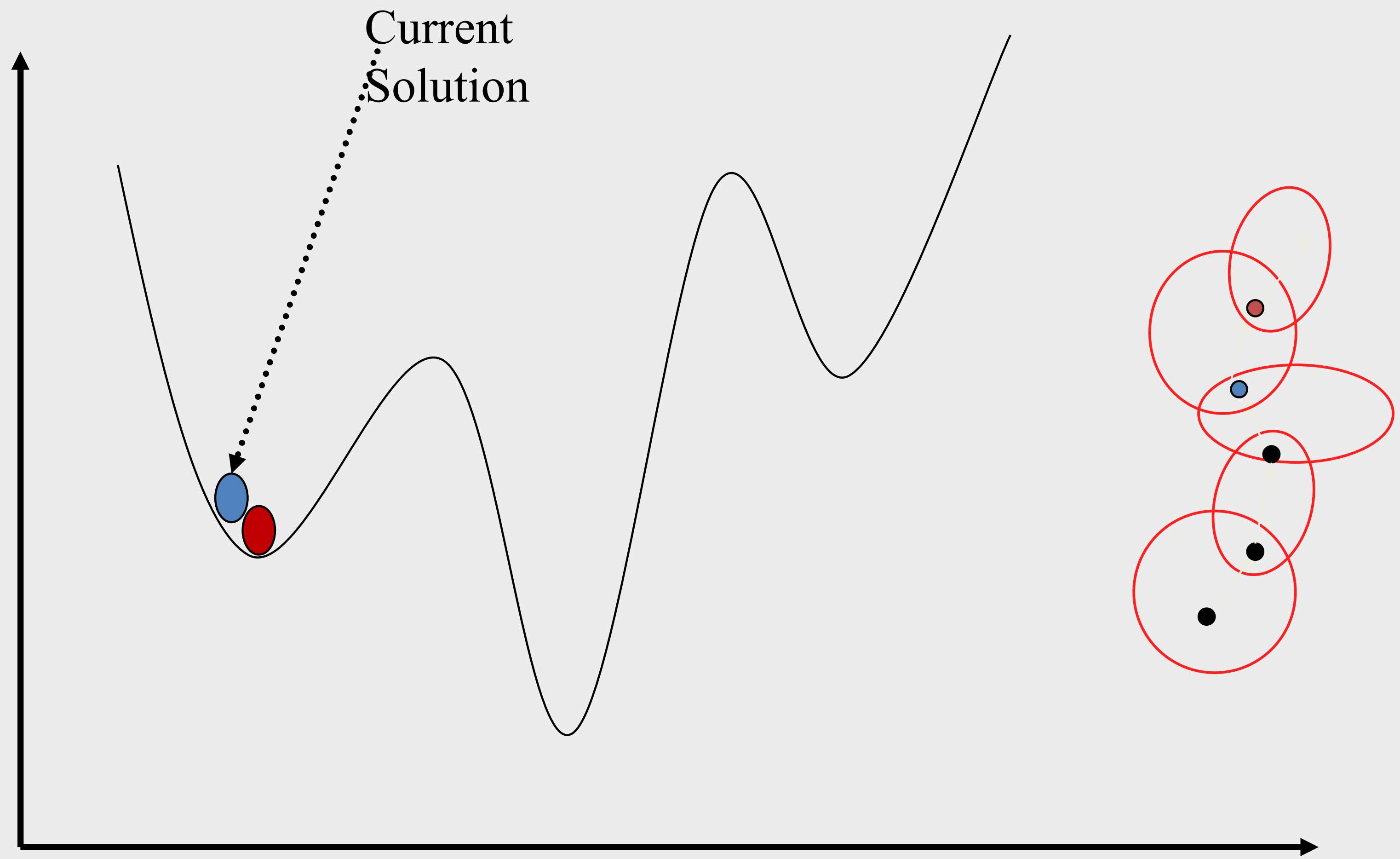




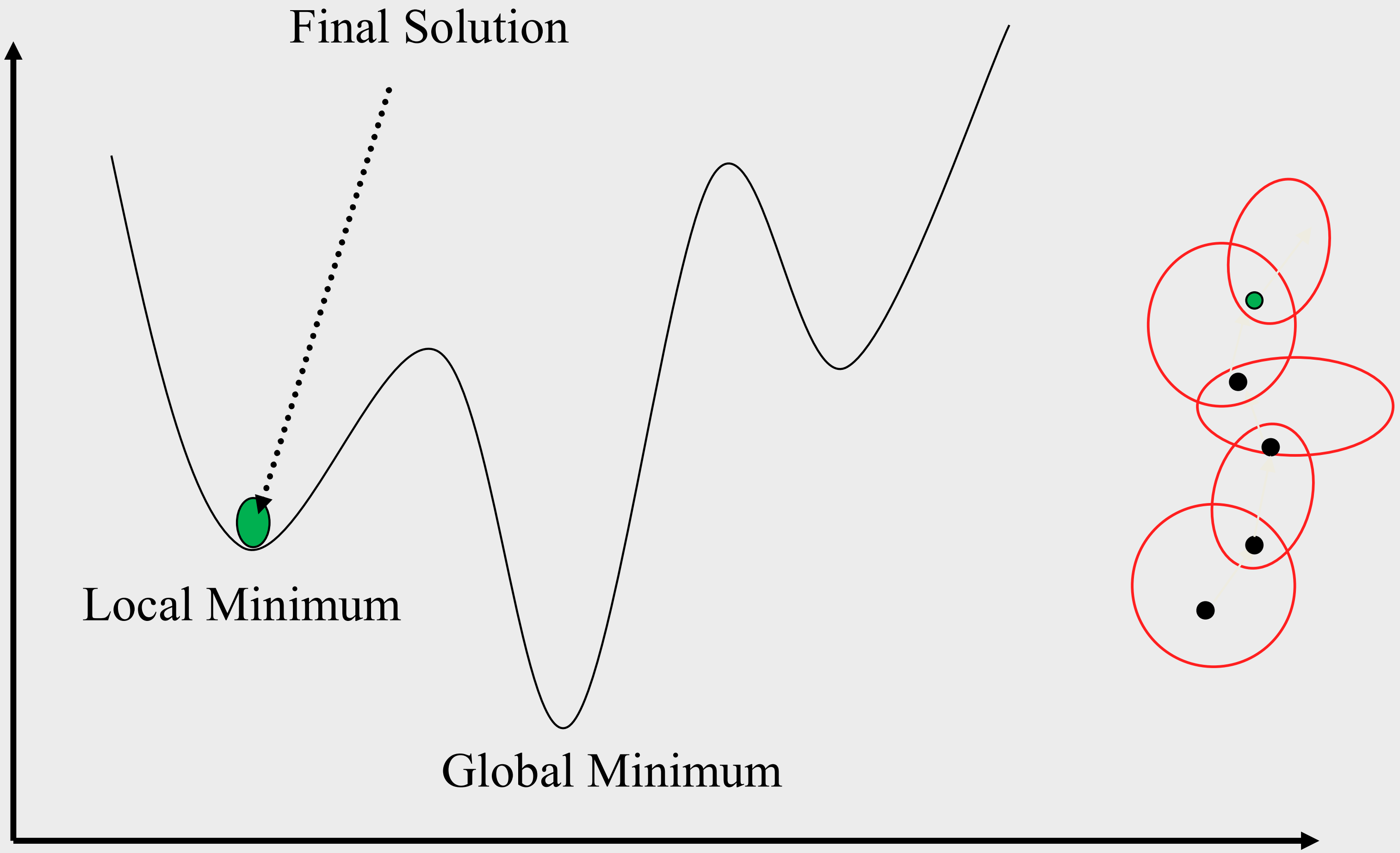
Local Search



Local Search

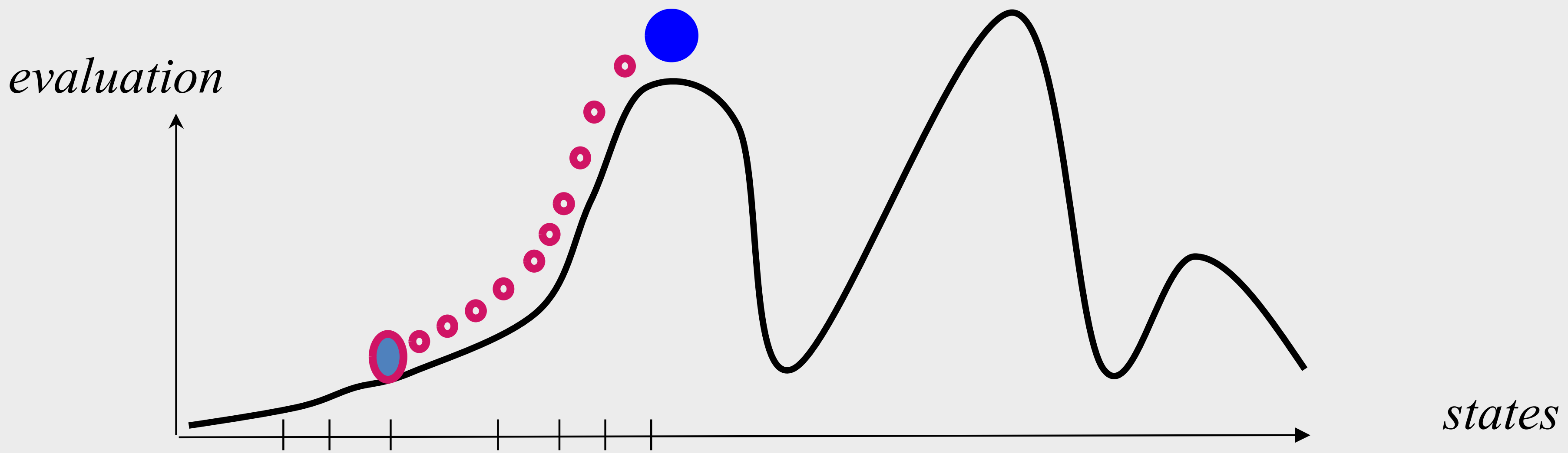


Local Search



LOCAL SEARCH – HILL CLIMBING

- Hill climbing search algorithm (also known as greedy local search) uses a loop that continually moves in the direction of increasing values (that is uphill).
- It terminates when it reaches a peak where no neighbor has a higher value.



- Local optimum:
  - If a solution x is "better" than all the solutions in its neighborhood, $N(x)$, we say that x is a local optimum
  - We note that local optimality is defined relative to a particular neighborhood

Local Search

In a local search we need the following:

- ✓ a Combinatorial Optimization Problem (COP)
- ✓ a starting solution (e.g. random)
- ✓ a defined search neighborhood (neighboring solutions)
- ✓ a move (e.g. changing a variable from $0 \rightarrow 1$ or $1 \rightarrow 0$), going from one solution to a neighboring solution
- ✓ a move evaluation function (a neighborhood evaluation)
- ✓ a move selection strategy
- ✓ a stopping criterion – e.g. a local optimum

Local Search–Disadvantages

- The search stops when no improvement can be found
- Restarting the search might help, but is often not very effective in itself
- Some neighborhoods can become very large (time consuming to examine all the neighbors)

Main Challenge in Local Search

How can we avoid the search stopping in a local optimum?

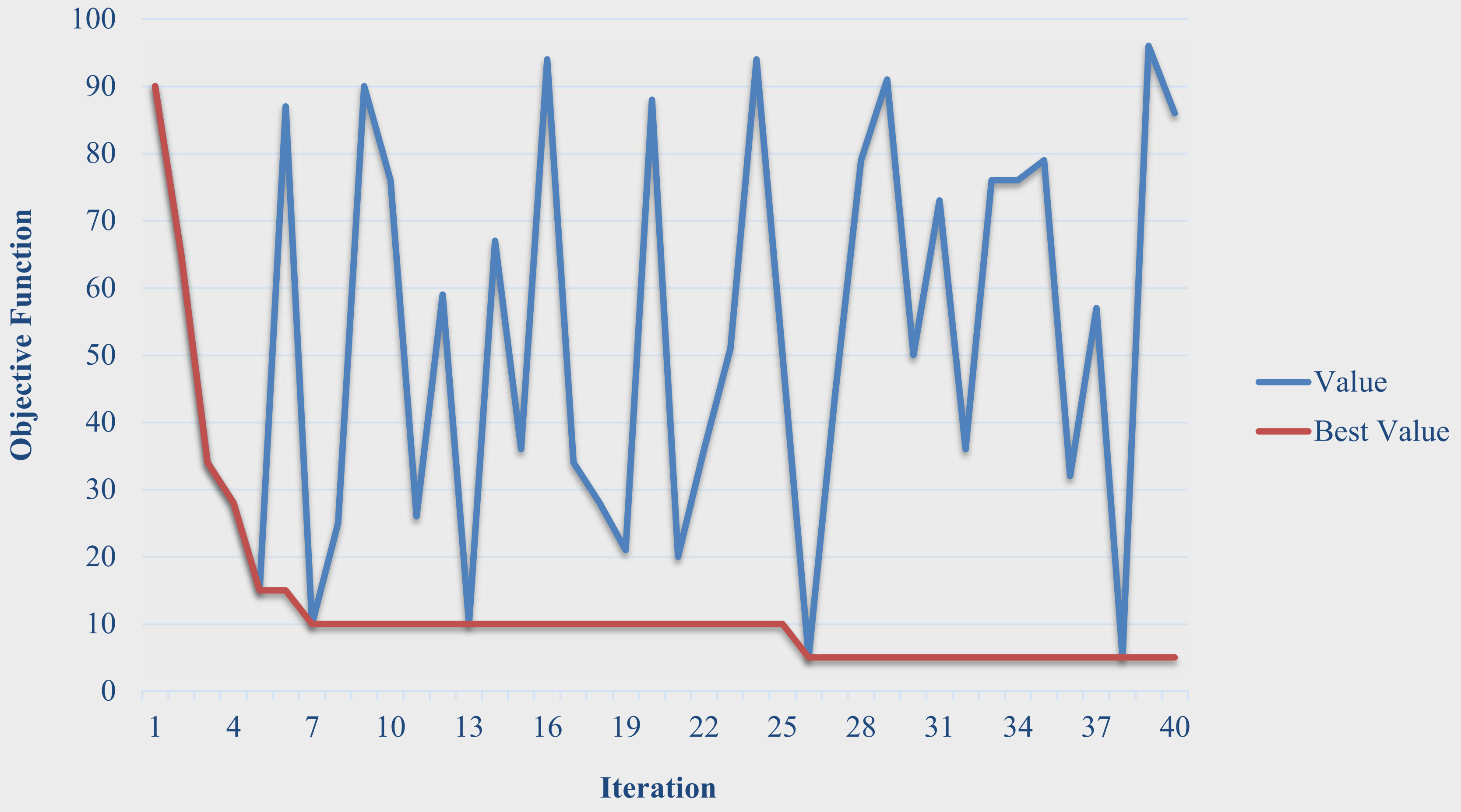
Diversification

How to Improve the Local Search

Some well-known Metaheuristics:

- Simulated Annealing (SA)
- Tabu Search (TS)
- Genetic Algorithms (GA)

Typical Search Trajectory



- In Local Search, we iteratively improve a solution by making small changes until we cannot make further improvements
- Metaheuristics can be used to guide a Local Search, and to help it to escape a local optimum
- Several metaheuristics are based on Local Search, but the mechanisms to escape local optima vary widely
  - We will look at Simulated Annealing and Tabu Search, as well as some others

LECTURE #6:

SIMULATED ANNEALING