# Optimization Methods

**1.** Introduction.
**2.** Greedy algorithms for combinatorial optimization.
**3. LS and neighborhood structures for combinatorial optimization.**
**4.** Variable neighborhood search, neighborhood descent, SA, TS.
**5**. Branch and bound algorithms, and subset selection algorithms.
**6.** Linear programming problem formulations and applications.
**7.** Linear programming algorithms.
**8.** Integer linear programming algorithms.
**9.** Unconstrained nonlinear optimization and gradient descent.
**10.** Newton's methods and Levenberg-Marquardt modification.
**11.** Quasi-Newton methods and conjugate direction methods.
**12.** Nonlinear optimization with equality constraints.
**13.** Nonlinear optimization with inequality constraints.
**14.** Problem formulation and concepts in multi-objective optimization.
**15.** Search for single final solution in multi-objective optimization.
**16:** Search for multiple solutions in multi-objective optimization.

**Travelling Salesman Problem (TSP):** **To find the shortest tour to visit all cities and return to the start city.**

**Input:** City set: $n$ cities ($i = 1, 2, ..., n$)

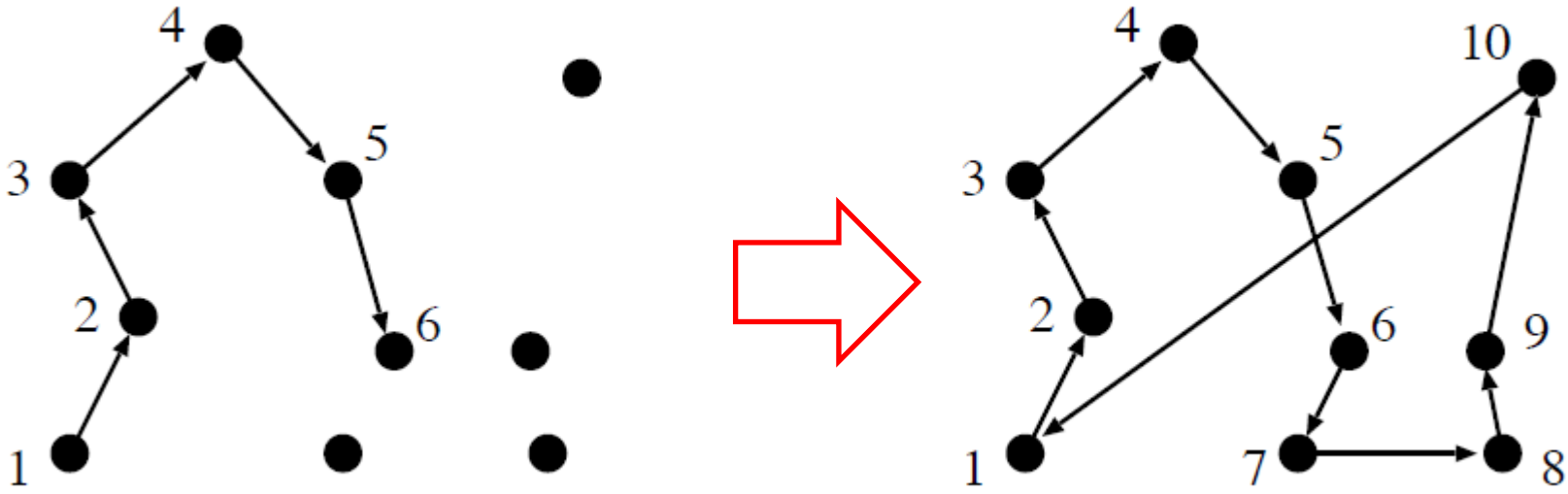
Distance between each pair of cities $(i, j)$: $d_{ij} = d_{ji}$

**Objective:** Minimization of a tour length starting from a city, visiting all cities and returning to the start city.

**Output:** Tour with the minimum distance

# Nearest Neighbor Greedy Method

1. Arbitrarily select a starting city.
2. Move from the current city to the nearest city among the remaining cities.

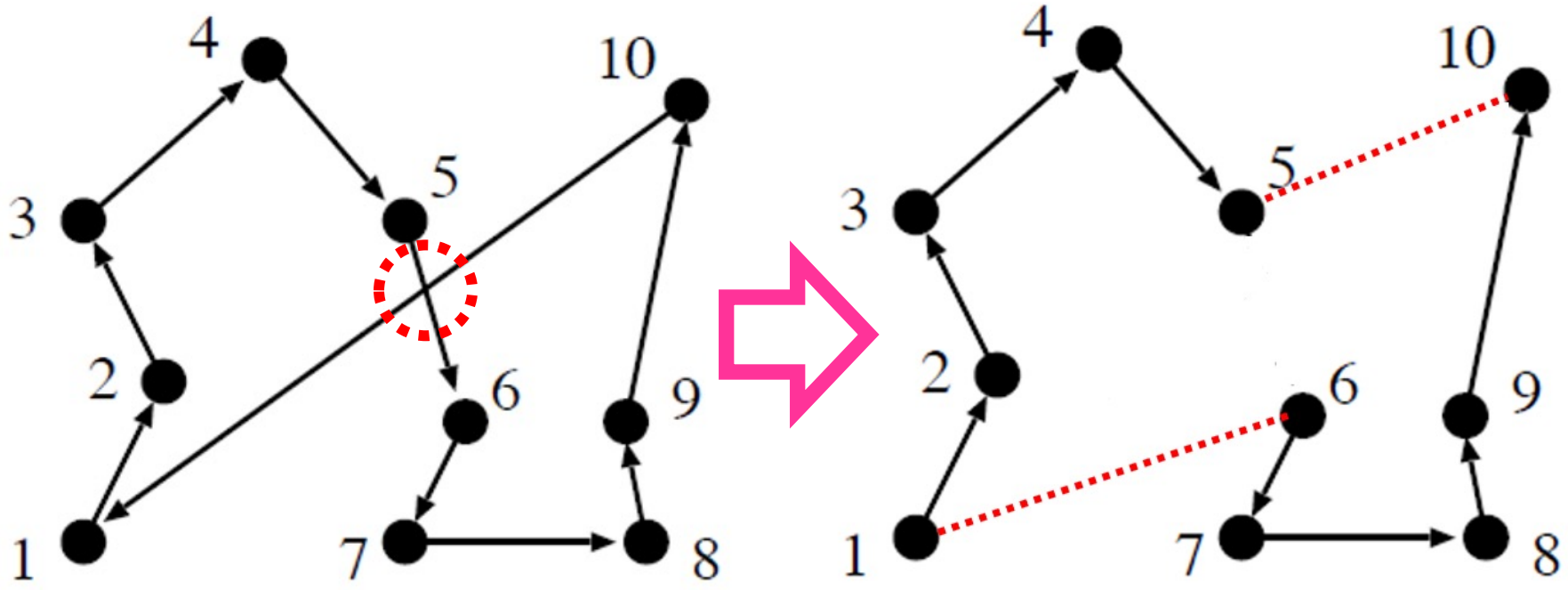



**In many cases, the obtained greedy solution is not optimal.**

# Greedy Algorithm:

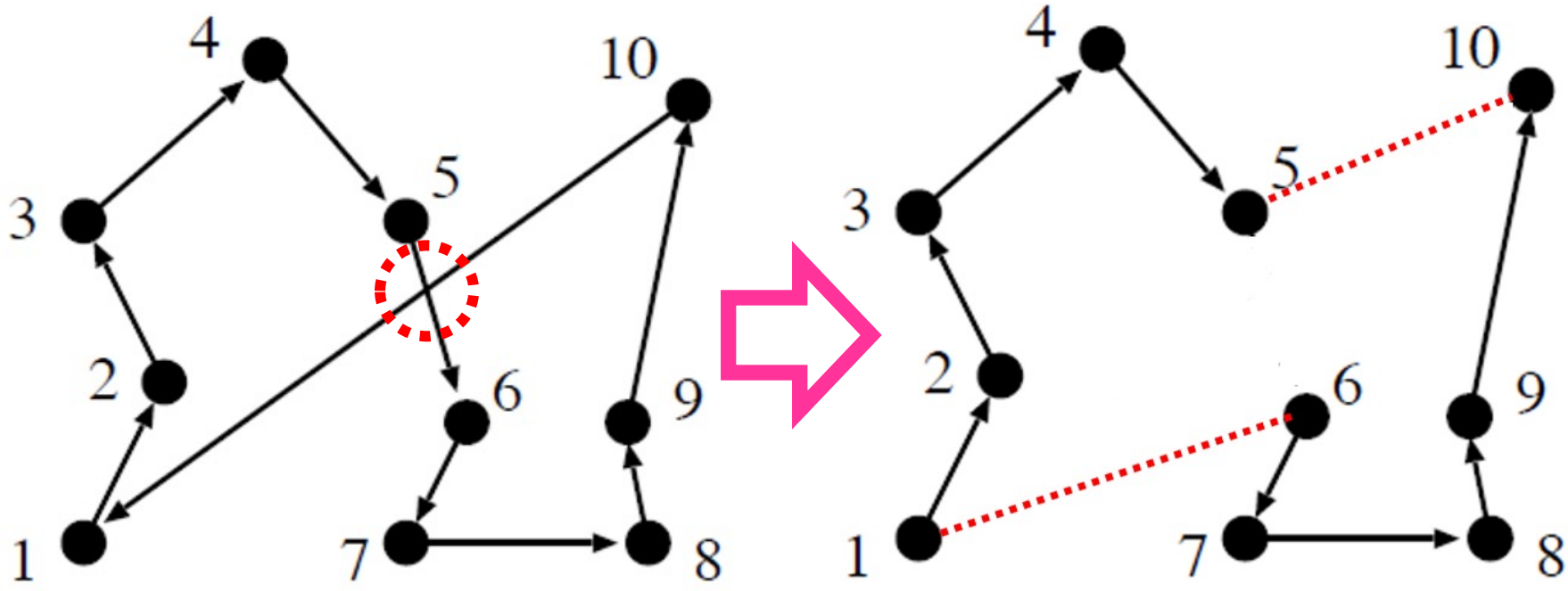
## A better tour can be obtained by small changes

In many cases, better tours can be obtained from the greedy tour **by small changes**. This is the main motivation for applying local search to the greedy tour.

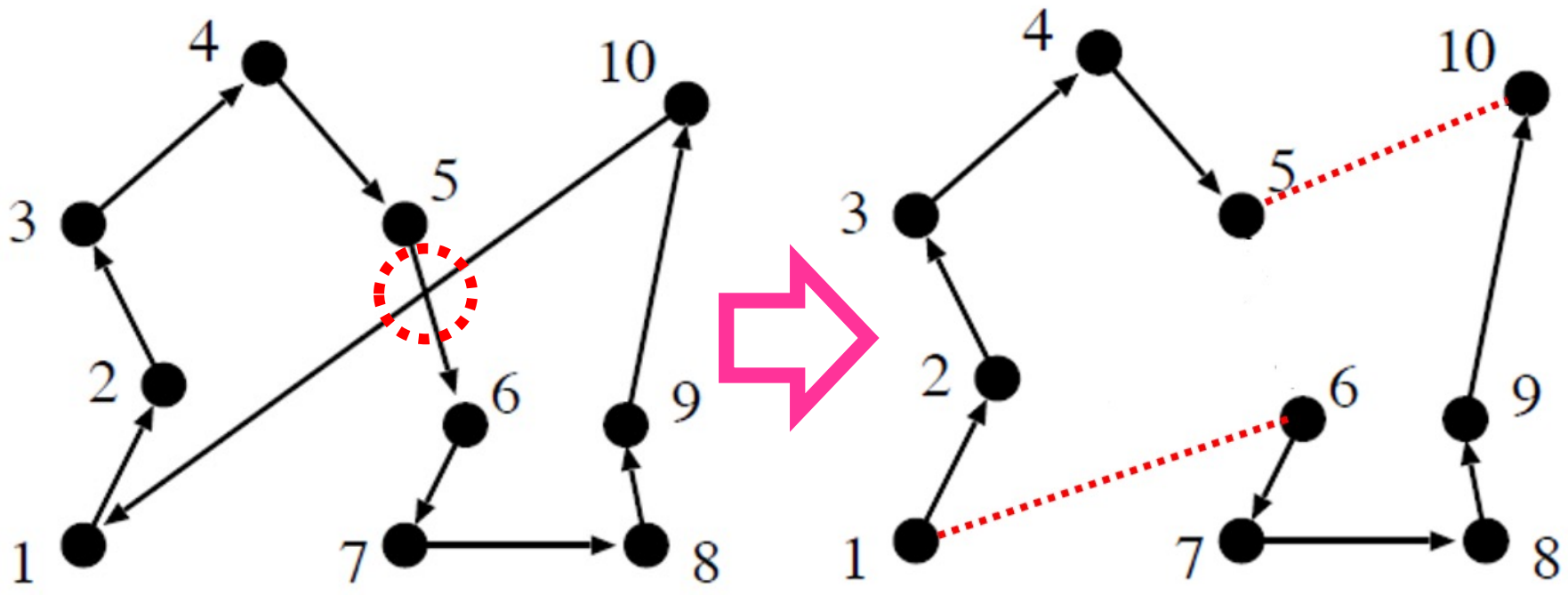


# The obtained tour is not optimal

If a tour has a cross-point (i.e., if two edges intersect with each other), it is not optimal. Geometrically it is clear in the following example that the modified tour with the dashed red lines is shorter than the original tour.



# To try to improve the current tour: Local Search (LS)




## LS: Basic Mechanism (iterative improvement):

1. Slightly modify the current tour.
2. If the current tour is improved, accept the modification.
   If the current tour is not improved, do not change it.
3. Go back to 1 for examining another modification.

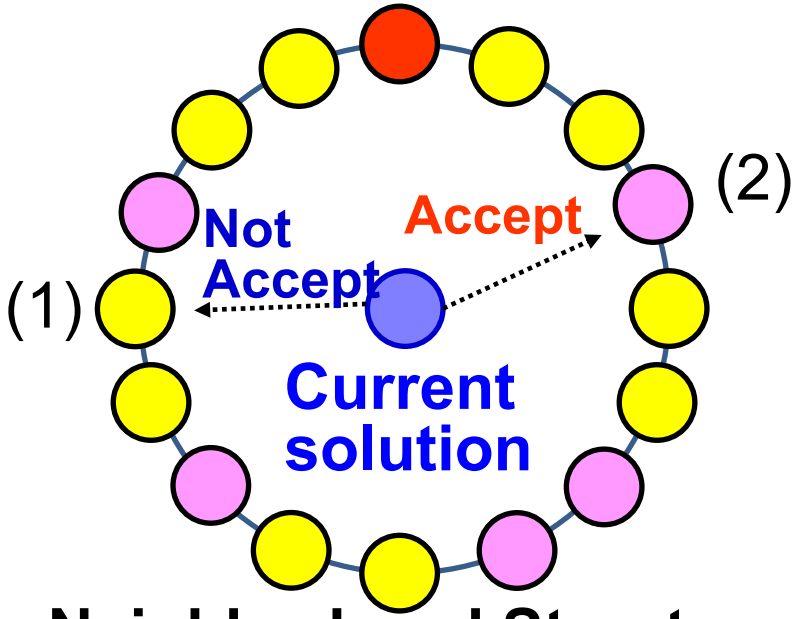
# Two Version of Local Search for Combinatorial Optimization

## 1. First Move Version

Examine neighboring solutions one by one. When a better solution is found, move to that solution (i.e., the first solution that is better than the current solution). If there is no better solution in the neighborhood, terminate the algorithm.

## 2. Best Move Version

Examine all neighboring solutions. When the best solution is better than the current solution, move to that solution (i.e., the best solution that is better than the current solution). If there is no better solution in the neighborhood, terminate the algorithm.




**Neighborhood Structure**

The best neighbor

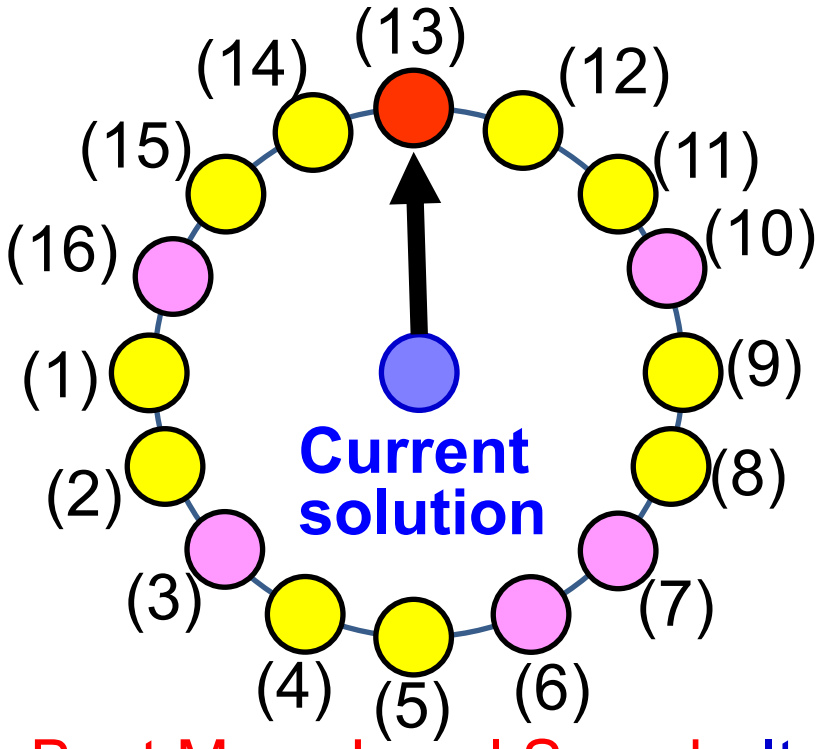
Better neighbor

Worse neighbor

First Move Local Search: Iterate the following procedure.

Examine **a new tour** generated by applying the inversion operation to the current tour just once. If the new tour is better than the current tour, the current tour is replaced **with the new tour**. Otherwise a different tour generated from the current tour by the inversion operation is examined. If all tours generated from the current tour are examined (i.e., if no new tour is better than the current tour), terminate the iteration of local search.

Examination order is important !




The best neighbor

Better neighbor

Worse neighbor

Best Move Local Search: Iterate the following procedure.

Examine **all new tours** generated by applying the inversion operation to the current tour just once. Then replace the current tour **with the best new tour** among all new tours (if the best new tour is better than the current tour). If the best new tour is not better than the current tour, terminate the iteration of local search.

Examination order is not important !

# Local Search (First Move Version)

1. Generate an initial solution $\boldsymbol{x}$.
2. Iterate the following steps:

(i) Generate a neighbor solution $\boldsymbol{y}$ of the current solution $\boldsymbol{x}$.

(ii) If $\boldsymbol{y}$ is better than $\boldsymbol{x}$, replace the current solution $\boldsymbol{x}$ with $\boldsymbol{y}$.

(iii) If no better solution exists in the neighborhood of $\boldsymbol{x}$, terminate the execution of the algorithm.

Local search is simple. However, it has a lot of implementation issues such as

- How to generate an initial solution $\boldsymbol{x}$ ==> **Greedy Solution**
- How to specify the neighborhood of the current solution $\boldsymbol{x}$,
- How to specify the order of neighbors to be examined

==> **Random Order**

# Local Search (Best Move Version)

1. Generate an initial solution ***x***.
2. Iterate the following steps:

(i) Find the best solution ***y*** in the neighborhood of the current solution ***x***.

(ii) If ***y*** is better than ***x***, replace the current solution ***x*** with ***y***. Otherwise terminate the execution of the algorithm.

Local search is simple. However, it has a lot of implementation issues such as

- How to generate an initial solution ***x*** ==> **Greedy Solution**
- How to specify the neighborhood of the current solution ***x***,
- ~~How to specify the order of neighbors to be examined~~

**Question: Which is better between the first move and the best move in local search?**

**Your answer: ?? move because . . .**

## 1. First Move Version

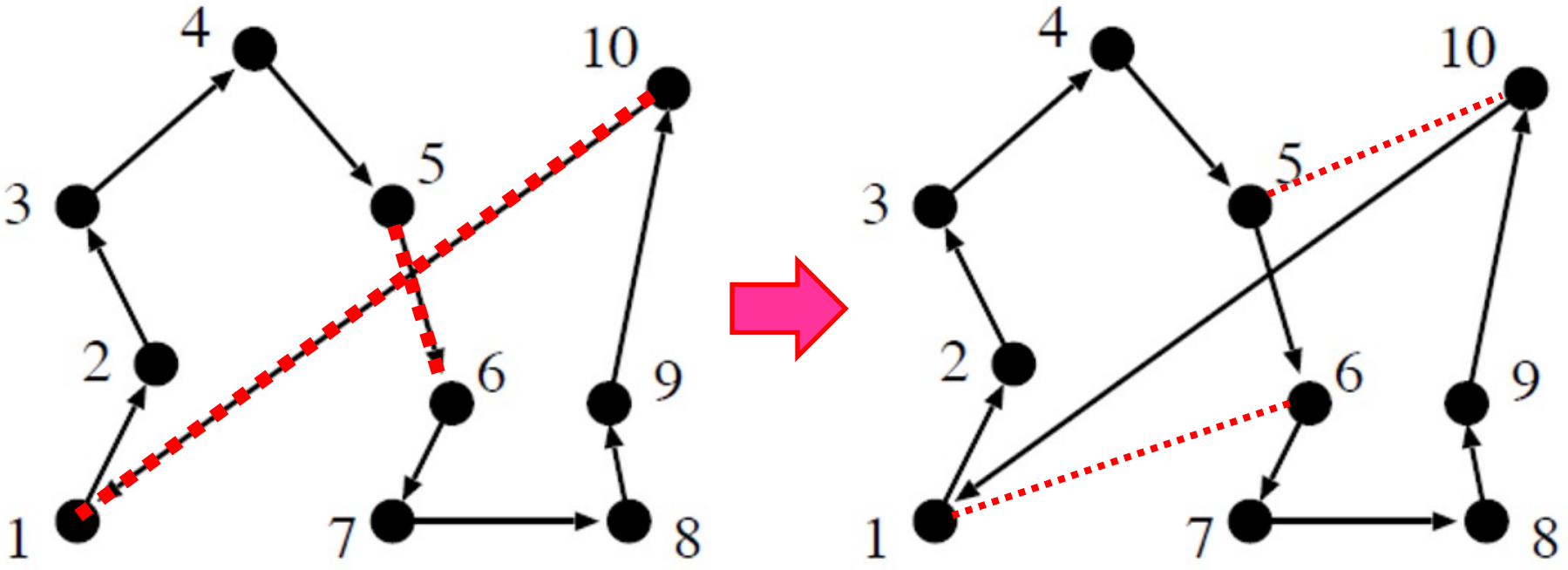
Examine neighboring solutions one by one. When a better solution is found, move to that solution (i.e., the first solution that is better than the current solution). If there is no better solution in the neighborhood, terminate the algorithm.

## 2. Best Move Version

Examine all neighboring solutions. When the best solution is better than the current solution, move to that solution (i.e., the best solution that is better than the current solution). If there is no better solution in the neighborhood, terminate the algorithm.

# The main issue in the design of Local Search

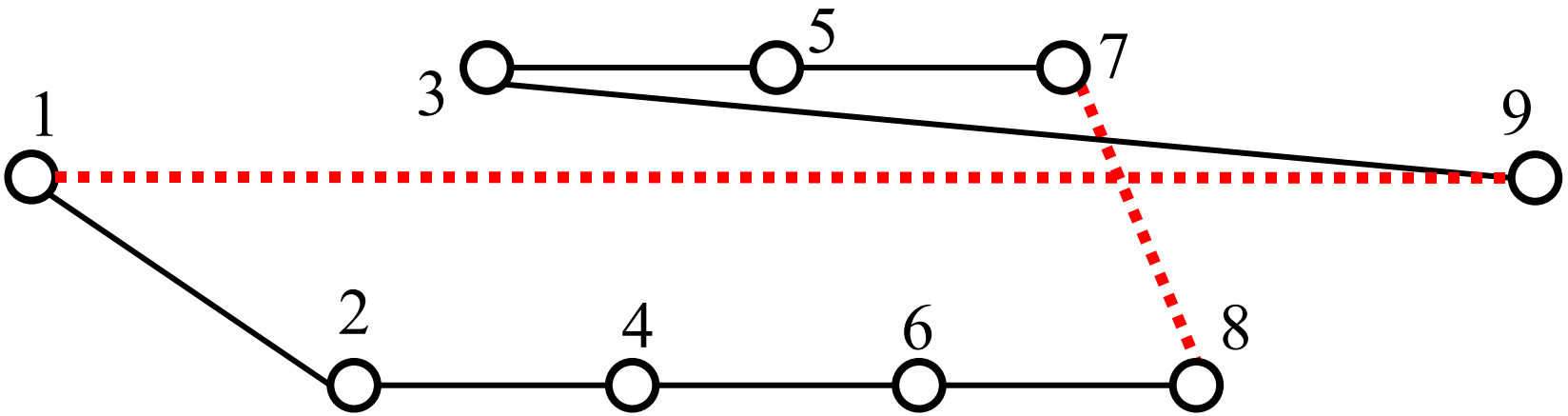
## How to specify the neighborhood of the current solution *x*



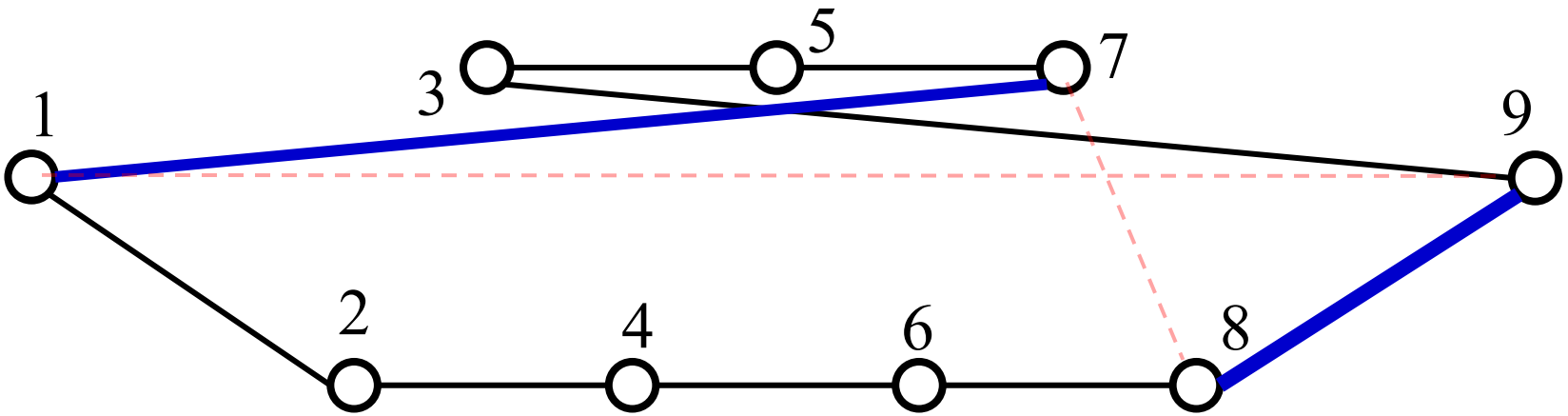

## Arbitrary Two-Edge Change

# The main issue in the design of Local Search

## How to specify the neighborhood of the current solution *x*

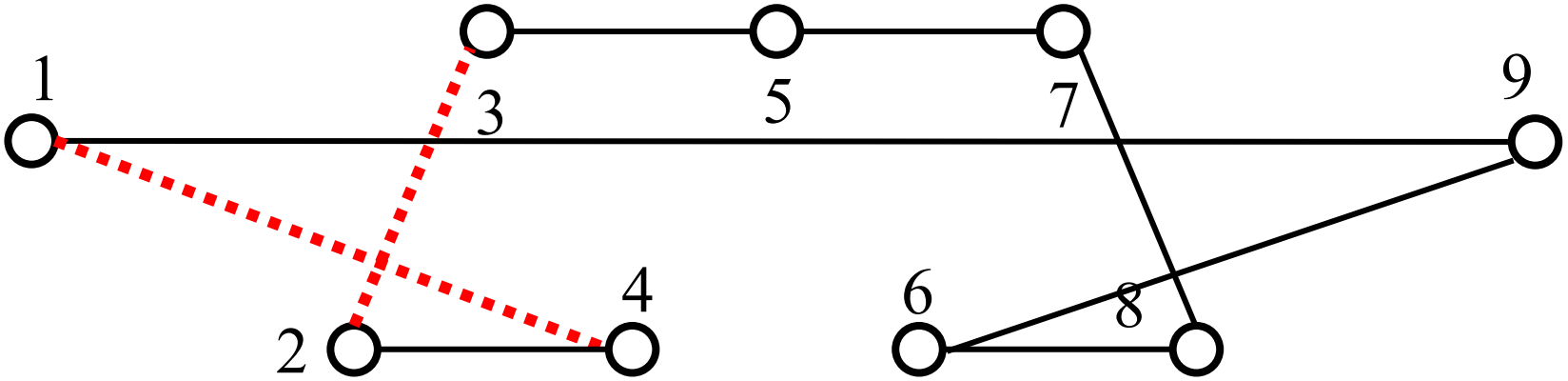



Arbitrary Two-Edge Change

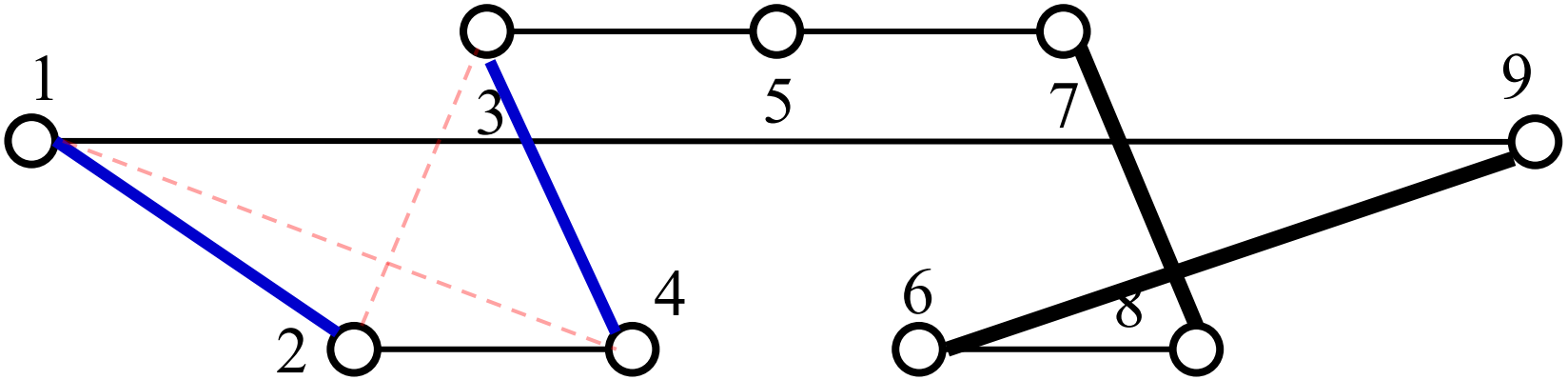


# The main issue in the design of Local Search

## How to specify the neighborhood of the current solution *x*

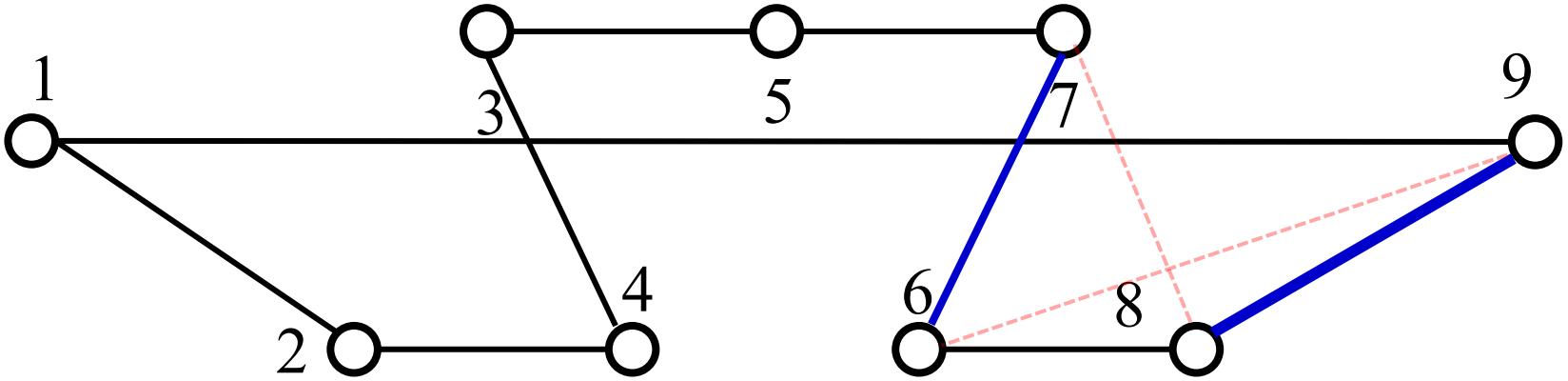



Arbitrary Two-Edge Change

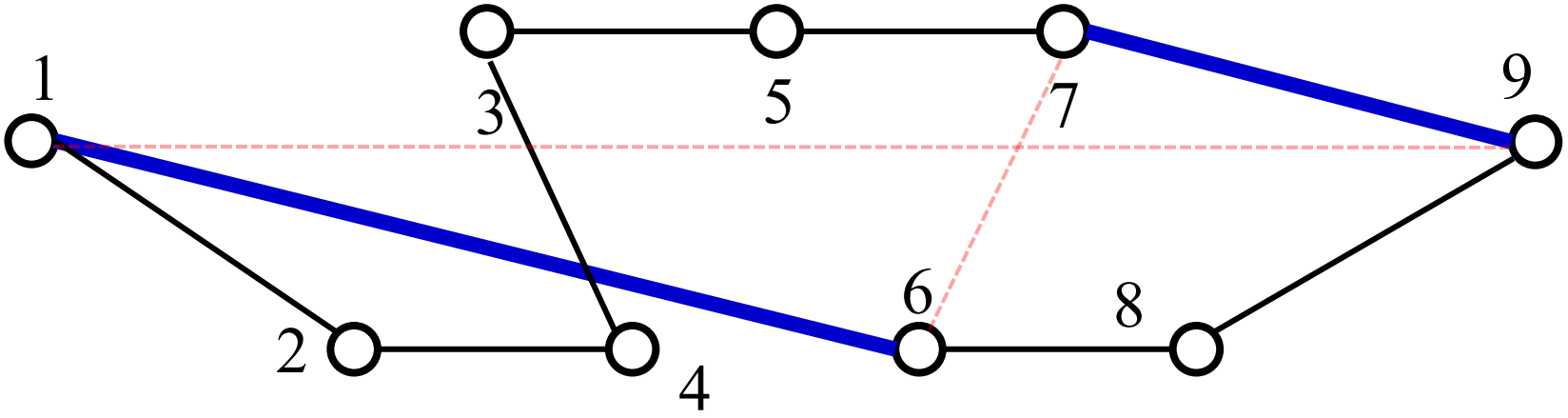


# The main issue in the design of Local Search

## How to specify the neighborhood of the current solution *x*

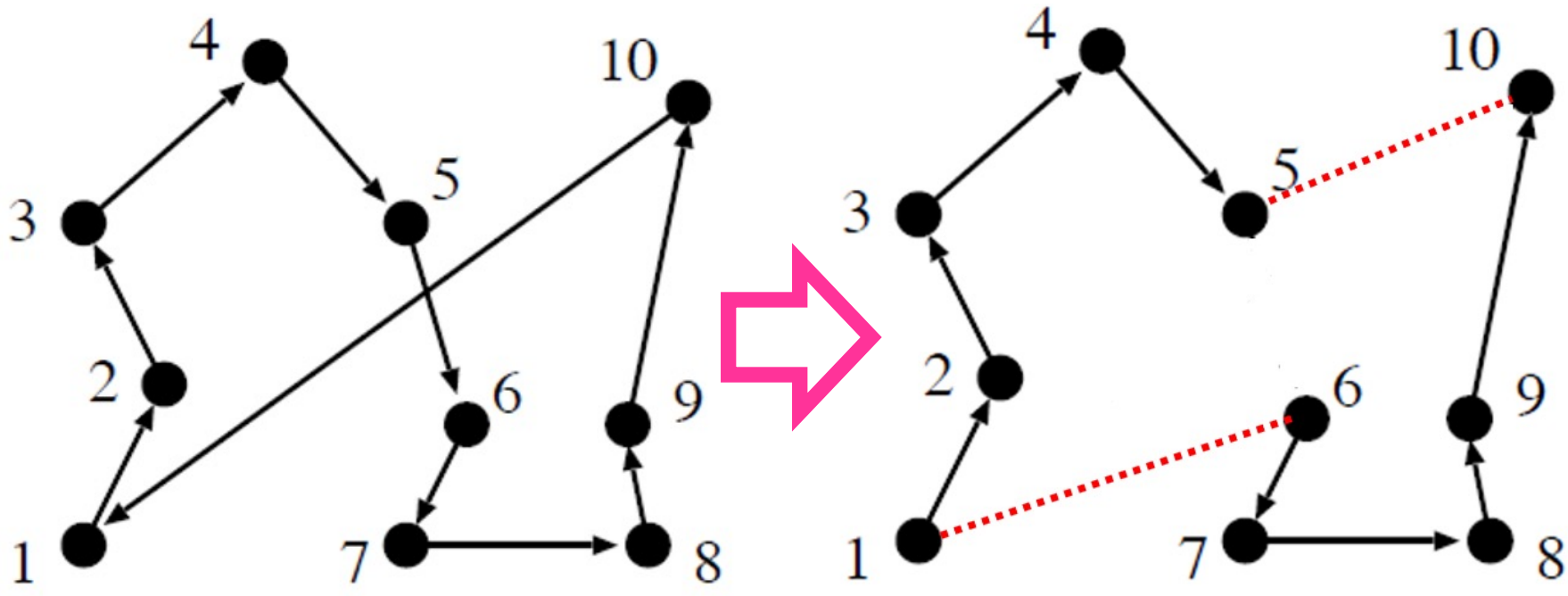



Arbitrary Two-Edge Change



# Local Search

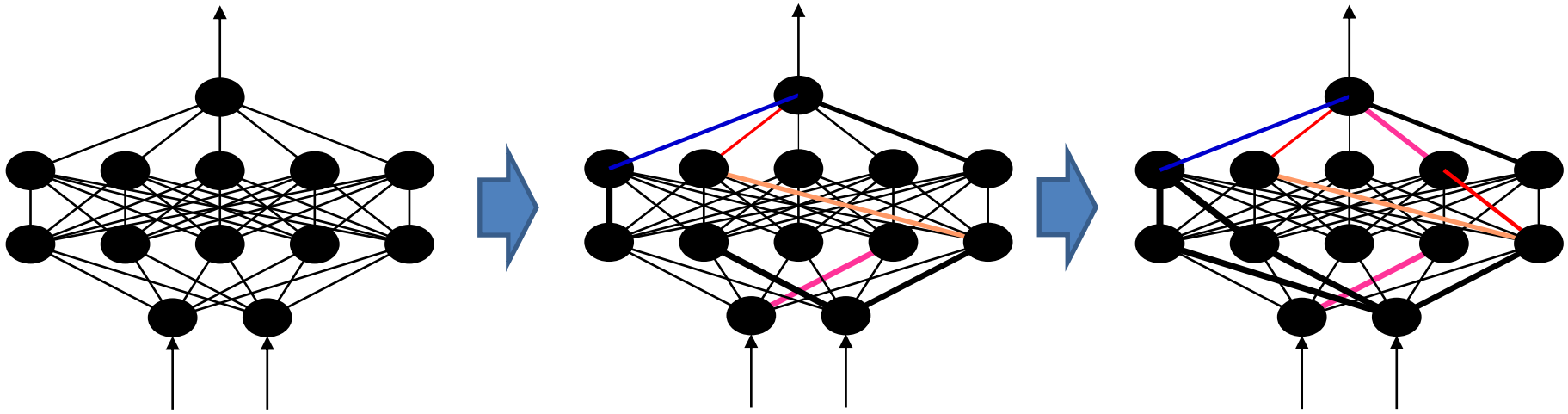
Local search is a very general concept which can be used in a wide variety of optimization problems. The basic idea is to move to a better solution after examining the neighborhood of the current solution (i.e., to move to its better neighbor). If there is no better neighbor, local search is terminated.



# Local Search

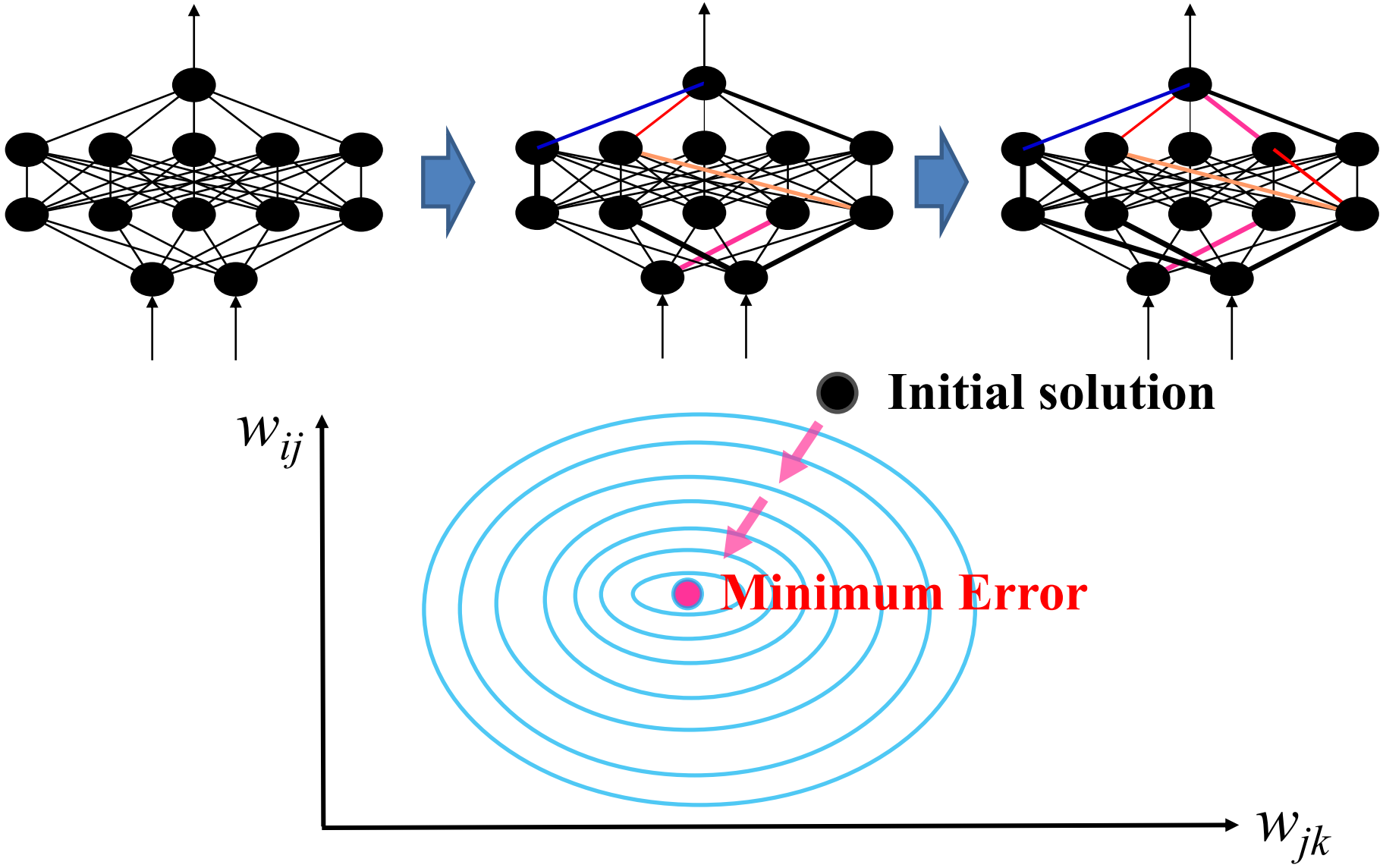
Local search is a very general concept which can be used in a wide variety of optimization problems. For example, the back-propagation (BP) algorithm of neural networks can be viewed as a local search algorithm for non-linear continuous optimization.

Minimization of $f(\boldsymbol{w})$ $\Rightarrow$ $w_{ij} \Rightarrow w_{ij} - \alpha \dfrac{\partial f(\boldsymbol{w})}{\partial w_{ij}}$



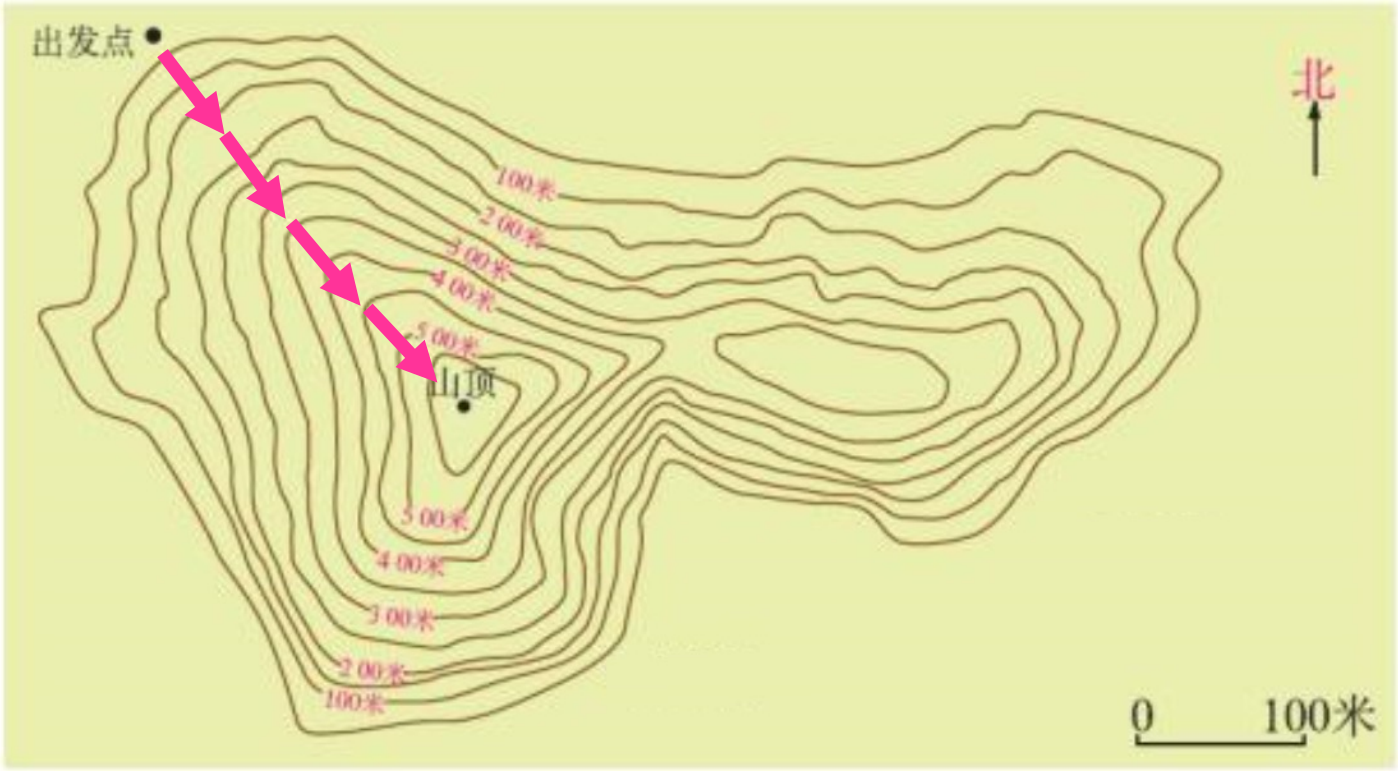
# Local Search

Back-Propagation Algorithm



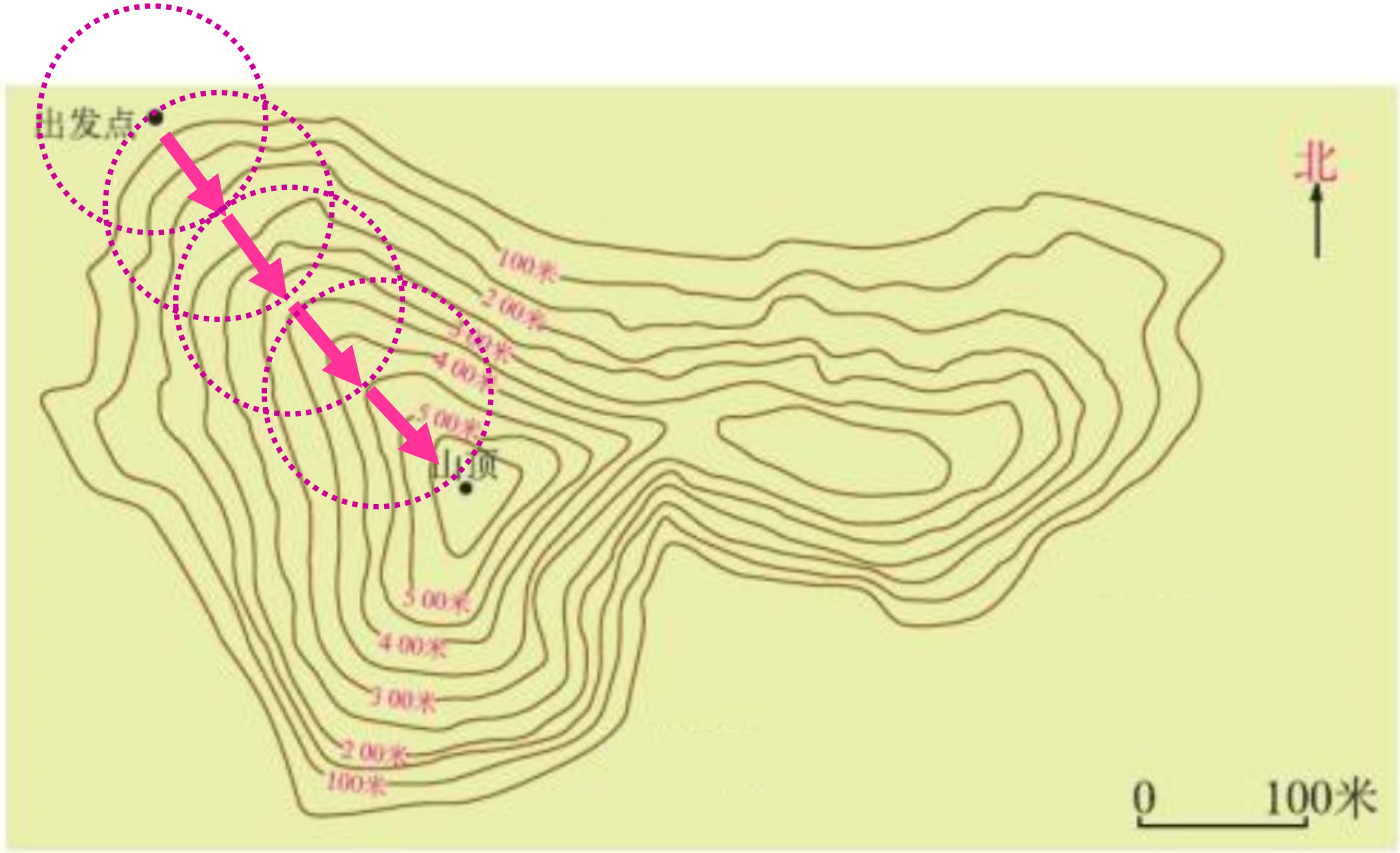
# Local Search

Local search is also called "**hill climbing**" for continuous maximization problems.



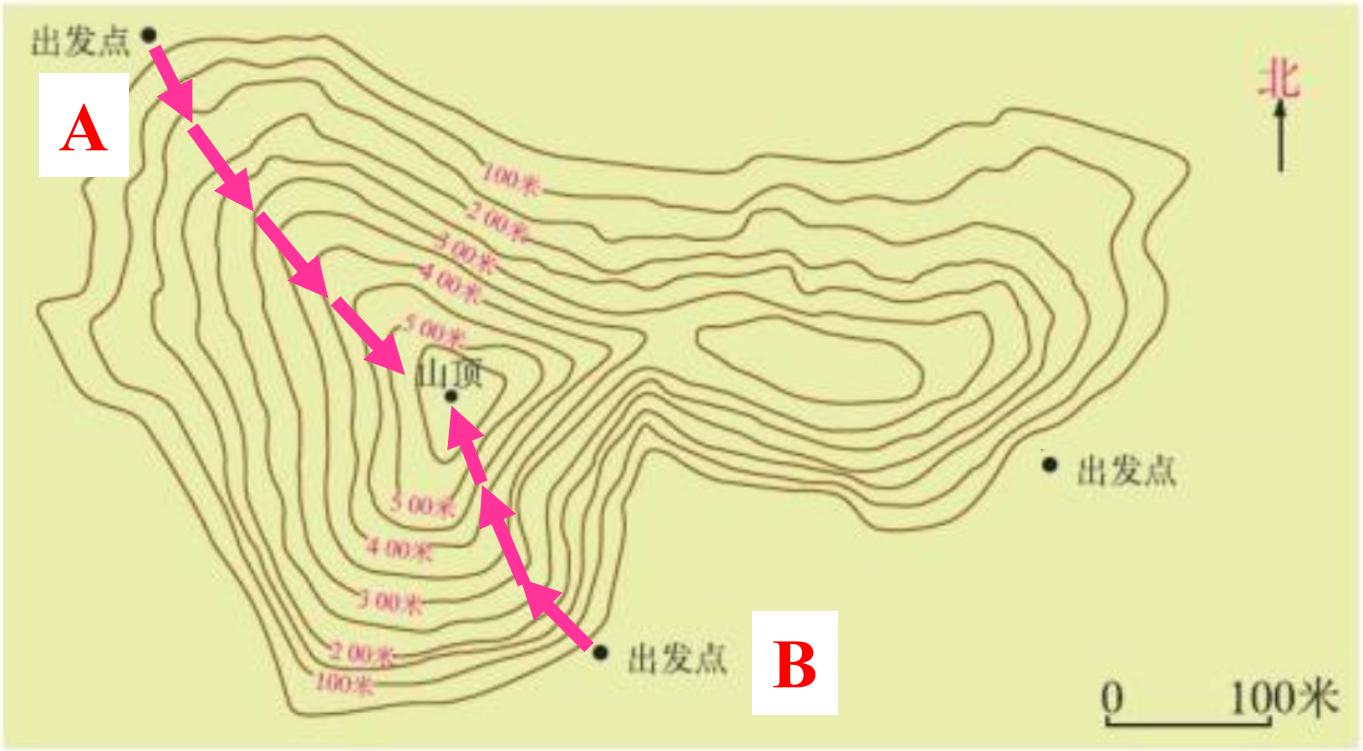
# Local Search: Advantages

**Efficient search and easy implementation:** Only a small region of the current solution is examined (efficient search). The current solution is replaced with a better neighbor. If the current solution has no better neighbor, the local search is terminated (simple implementation).



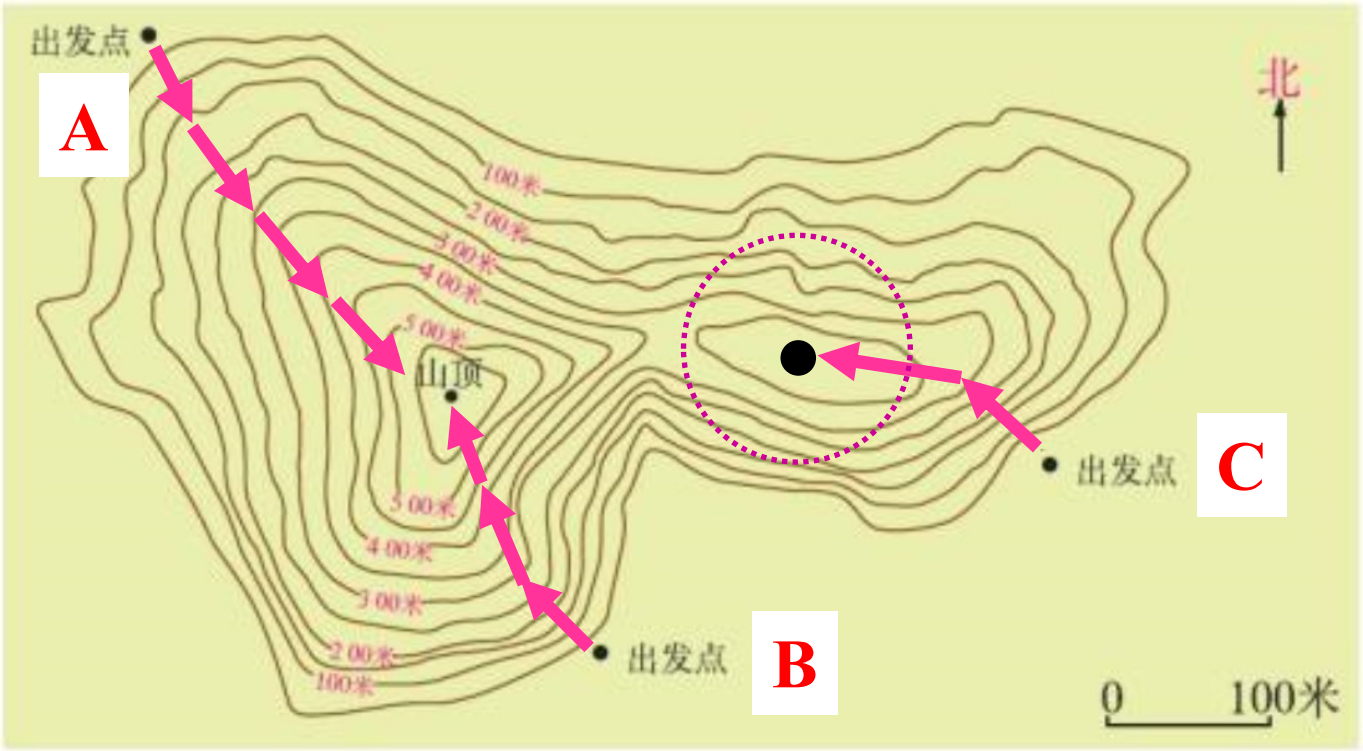
# Local Search: Disadvantages

**Local optimality and dependency on the initial solution:** Local search is terminated when no better solutions are included in the neighborhood of the current solution. Thus the obtained solution is a locally optimal solution. The final solution totally depends on the choice of an initial solution.



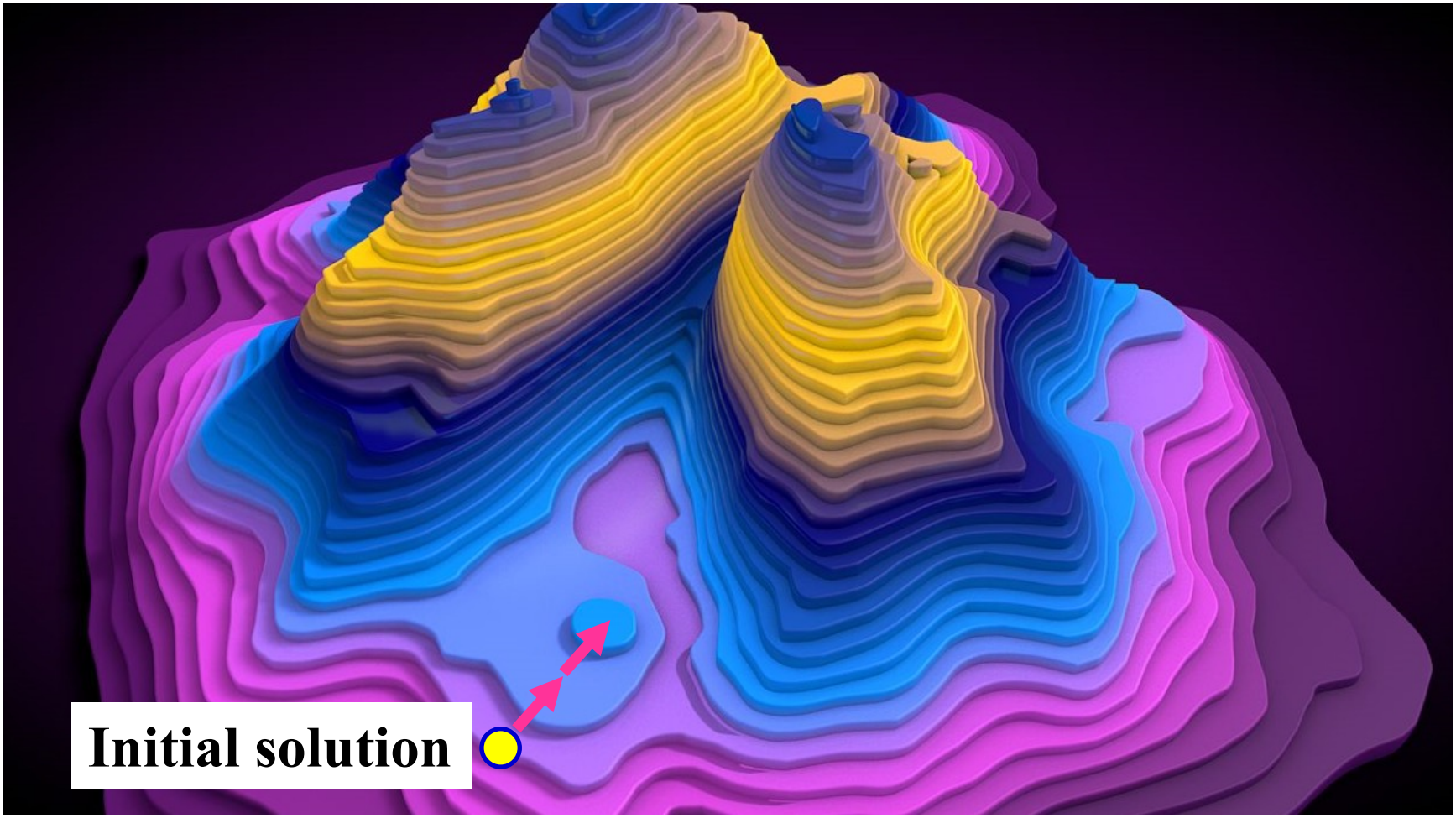
# Local Search: Disadvantages

**Local optimality and dependency on the initial solution:** Local search is terminated when no better solutions are included in the neighborhood of the current solution. Thus the obtained solution is a locally optimal solution. The final solution totally depends on the choice of an initial solution.



# Local Search: Disadvantages

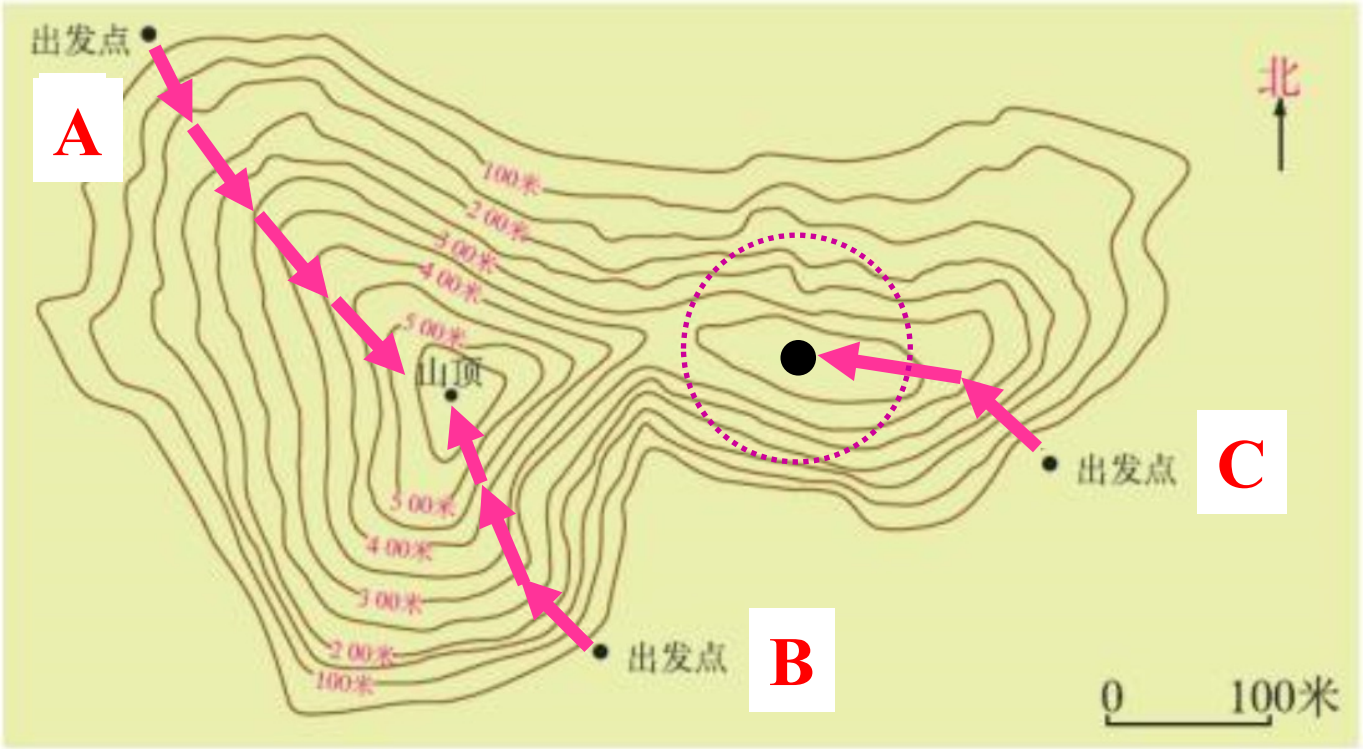
**Local optimality and dependency on the initial solution:**
In some cases, the quality of the final solution can be very poor.



# Handling of Local Optimality:

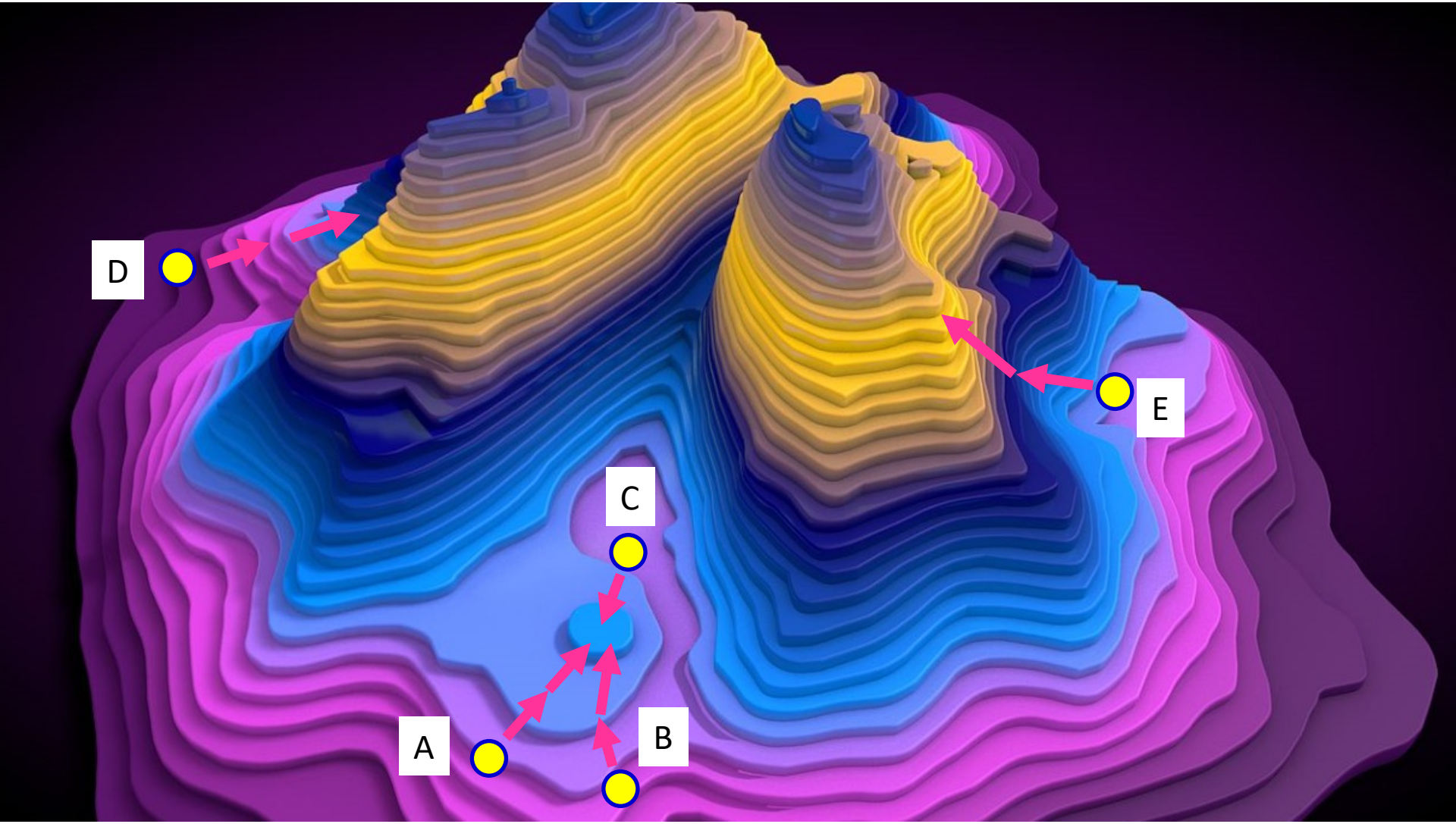
## (i) Use of many initial solutions

Roughly speaking, the final solution of local search is the nearest local solution from the initial solution. It is likely that the optimal solution (or good local solution) can be obtained by using well-distributed initial solutions.



# Handling of Local Optimality:

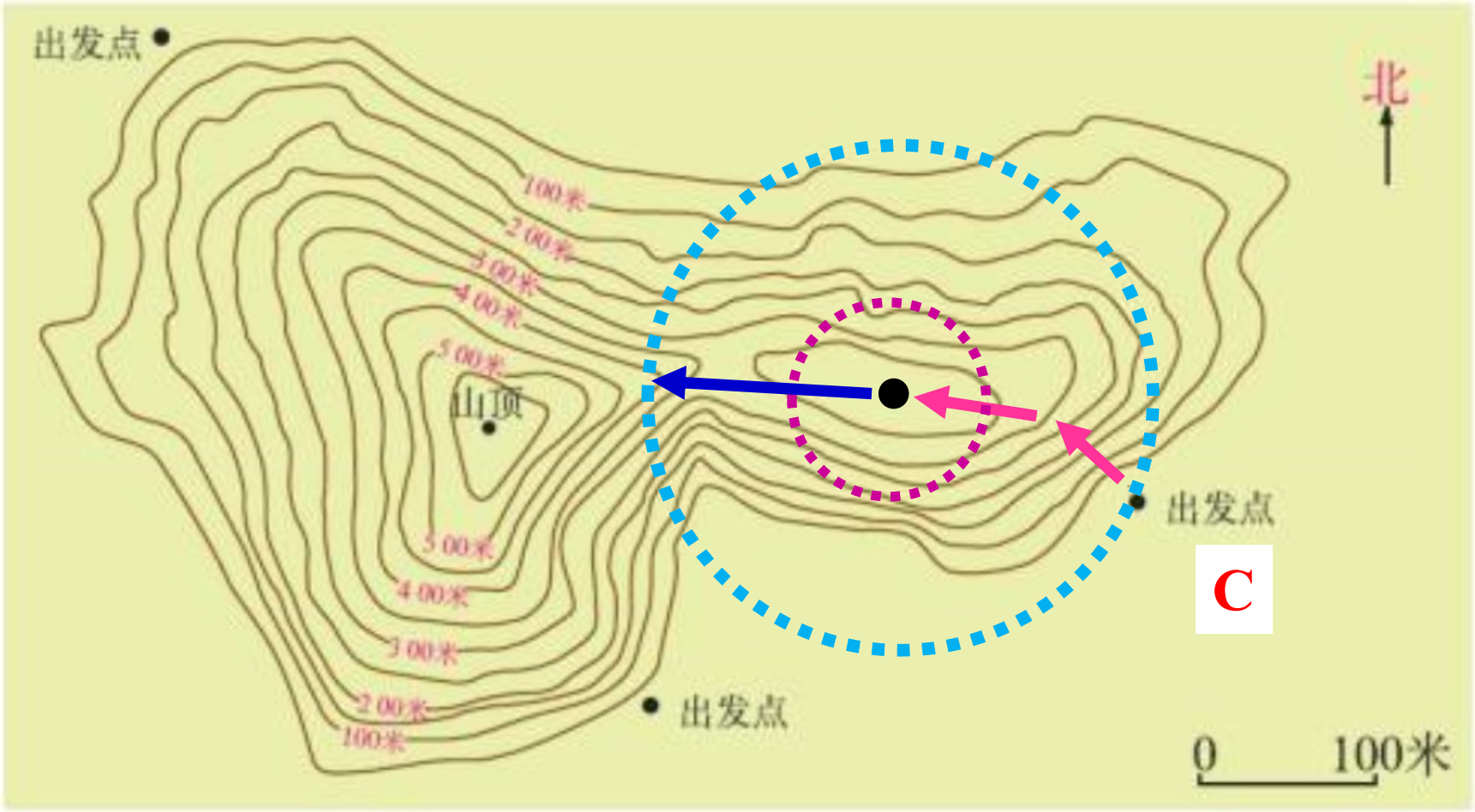
## (i) Use of many initial solutions



# Handling of Local Optimality:

## (ii) Use of a large neighborhood

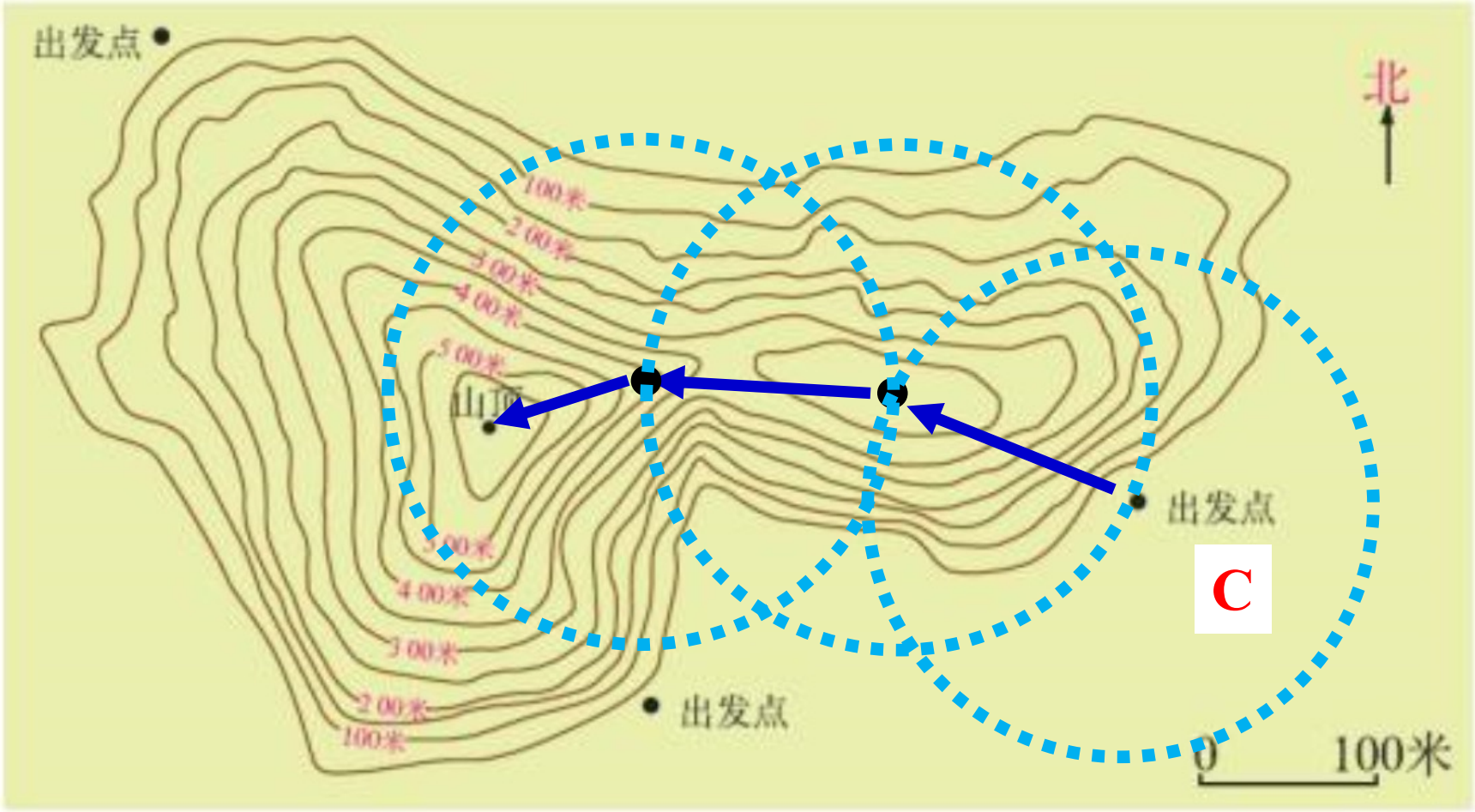
By using a much larger neighborhood, we can escape from the local optimal solution.



# Handling of Local Optimality:

## (ii) Use of a large neighborhood

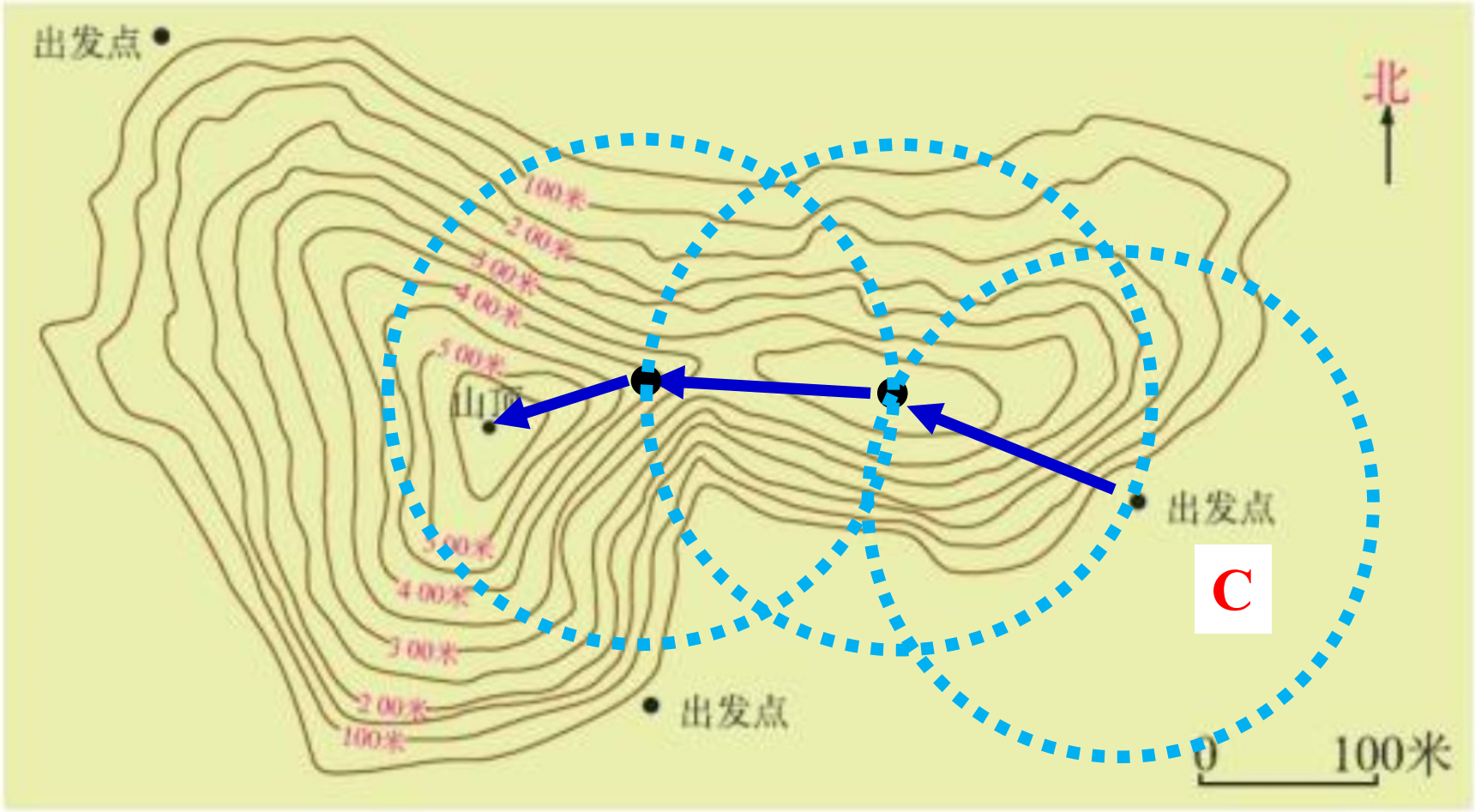
Then, we can find the optimal solution.



# Handling of Local Optimality:

## (ii) Use of a large neighborhood

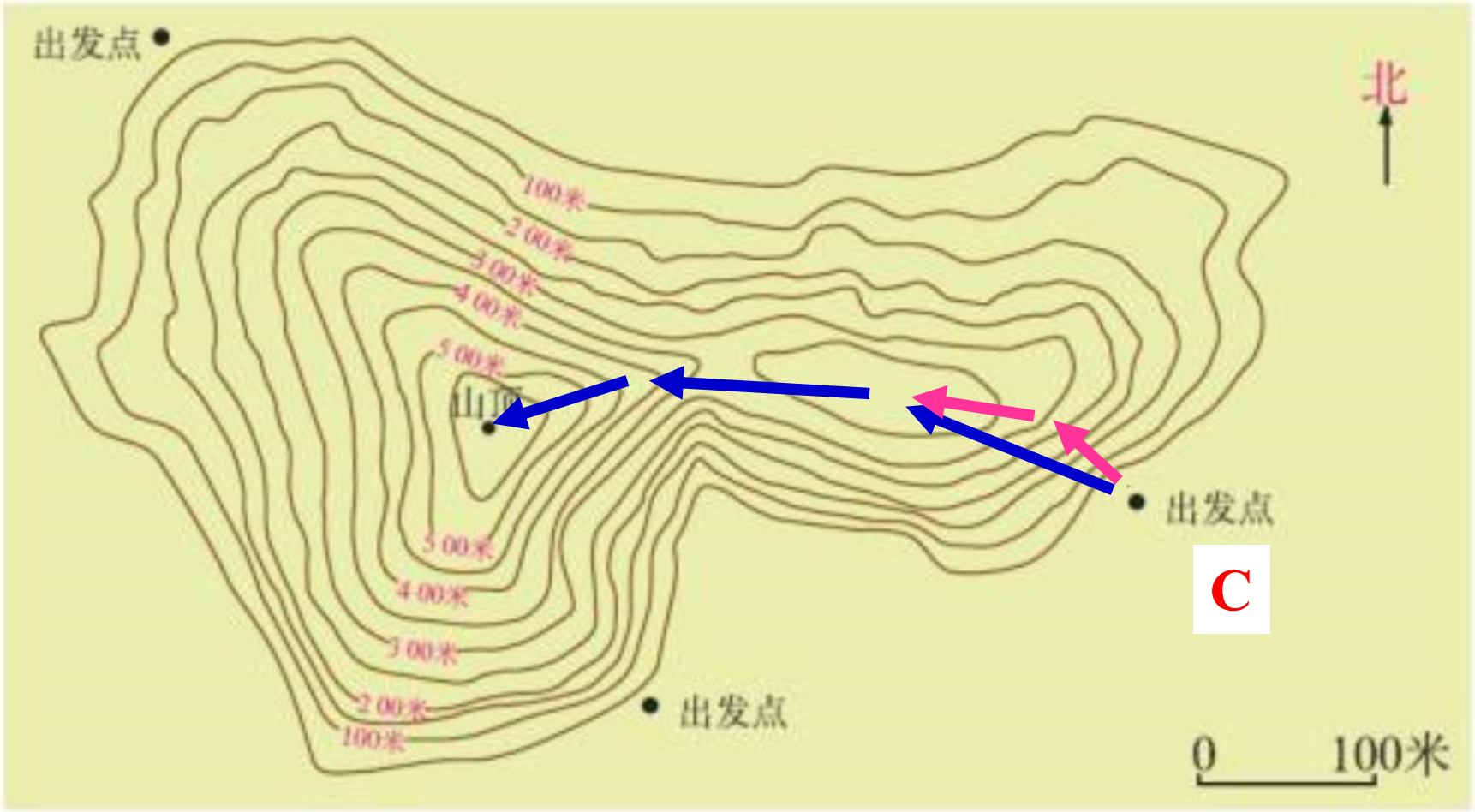
In this example, we can find the optimal solution from any initial solution with the neighborhood of the current size.



# Question:

## How to specify the neighborhood size.

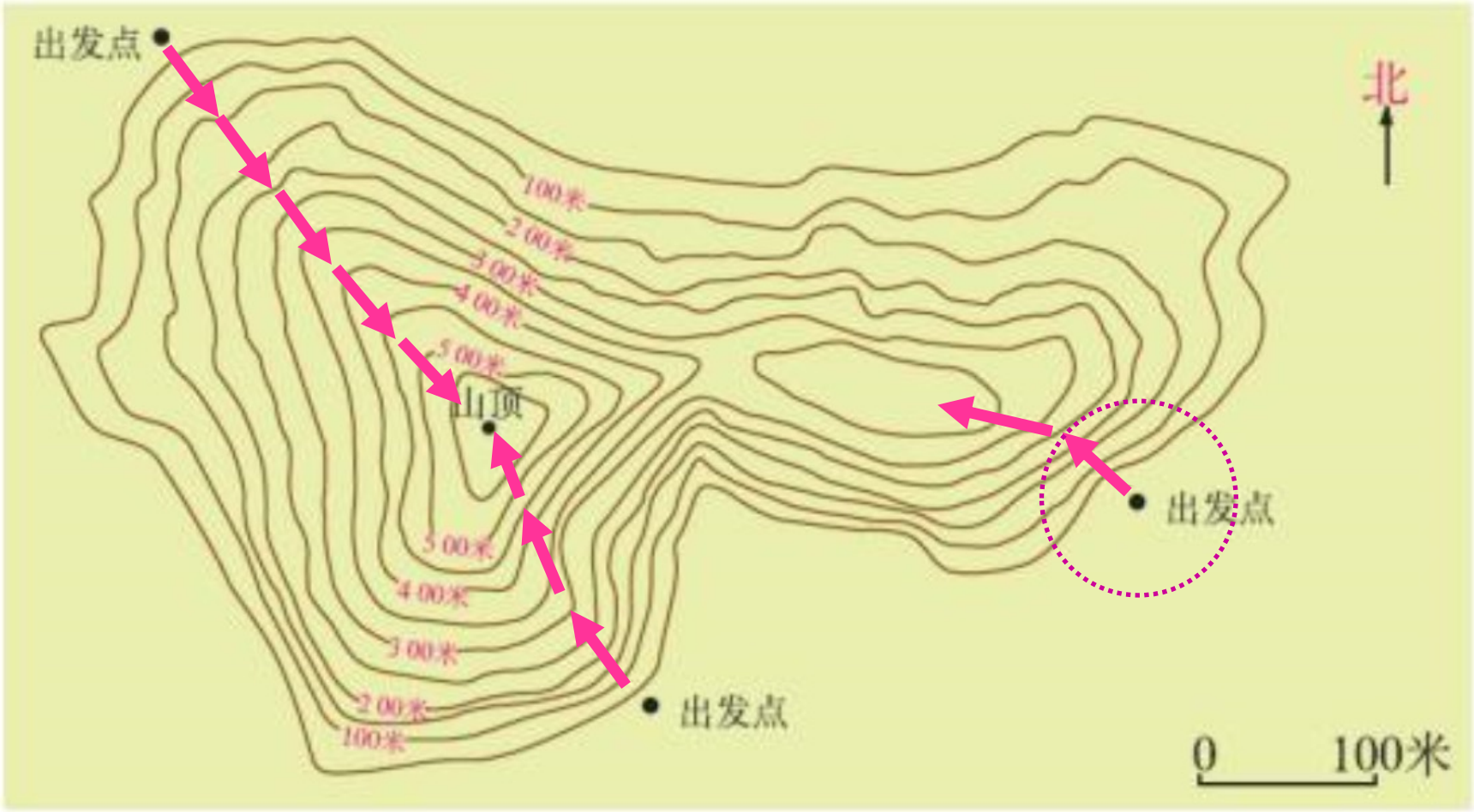
**Do you think that a large neighborhood is always good?**



# Small Neighborhood

**Advantages:_____.**

**Disadvantages:_____ .**

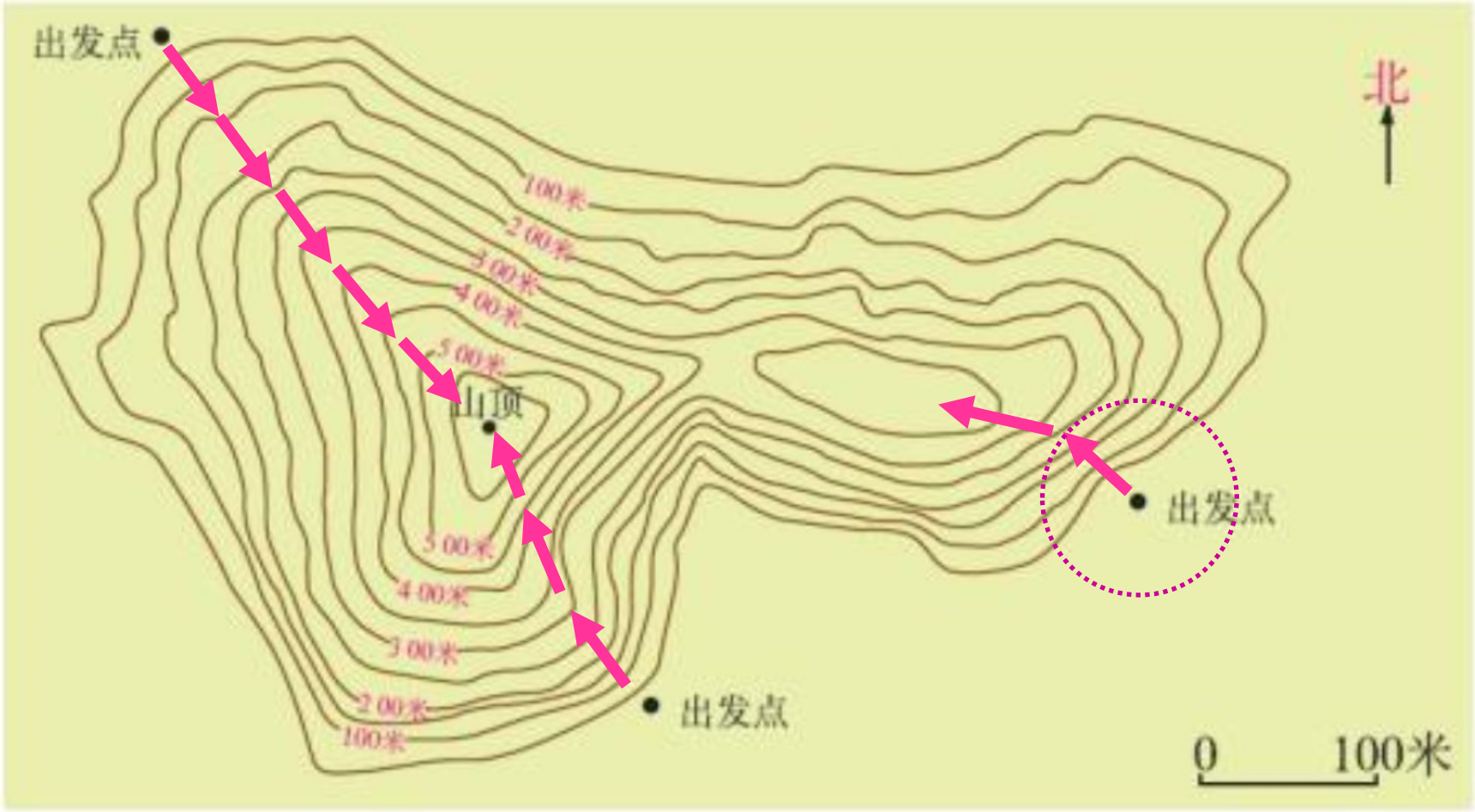


# Small Neighborhood

**Advantages:** Computation load is small for one step.

**Disadvantages:** It is not easy to find the global solution.

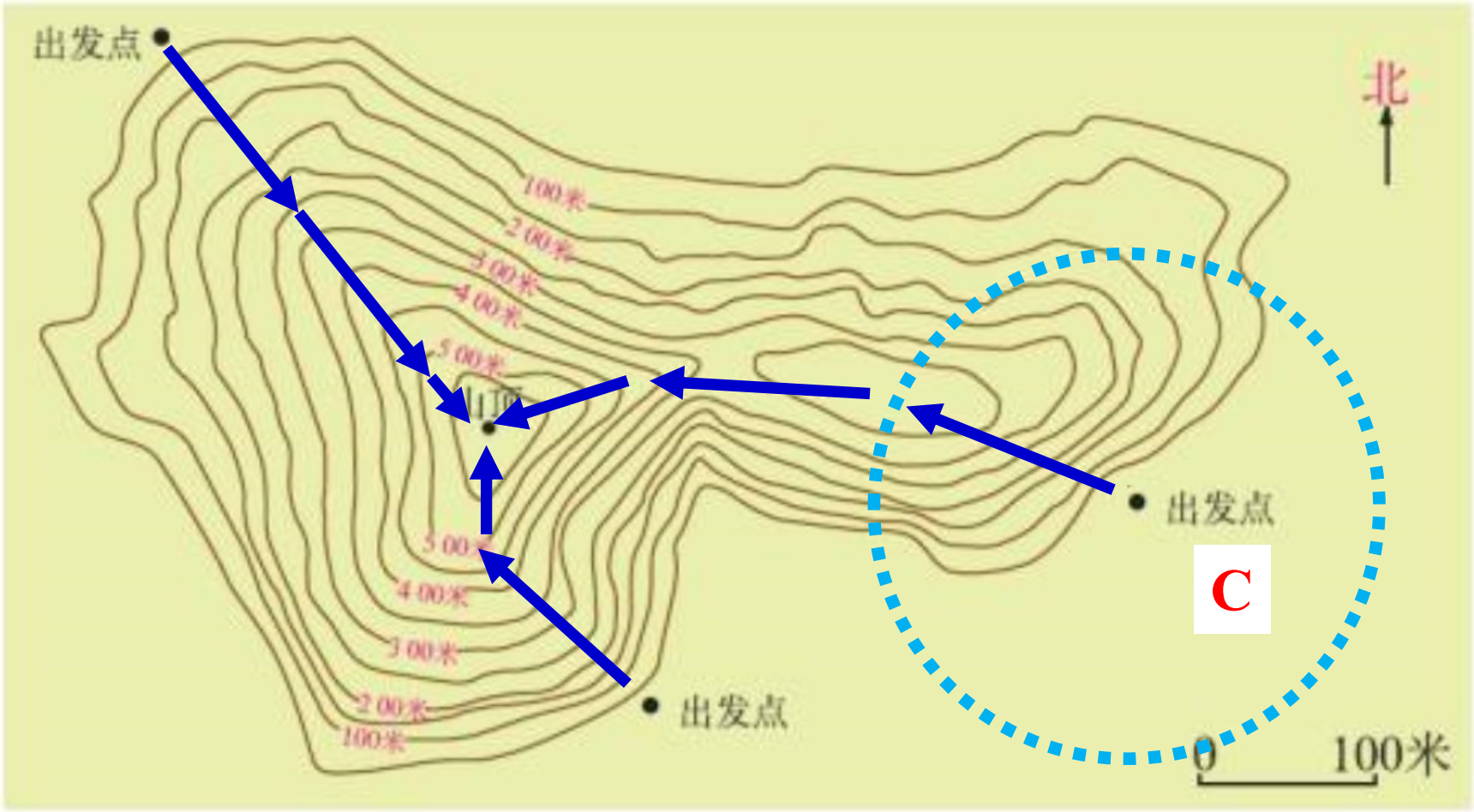
It may need many steps to find a good solution.



# Large Neighborhood

**Advantages:_____.**

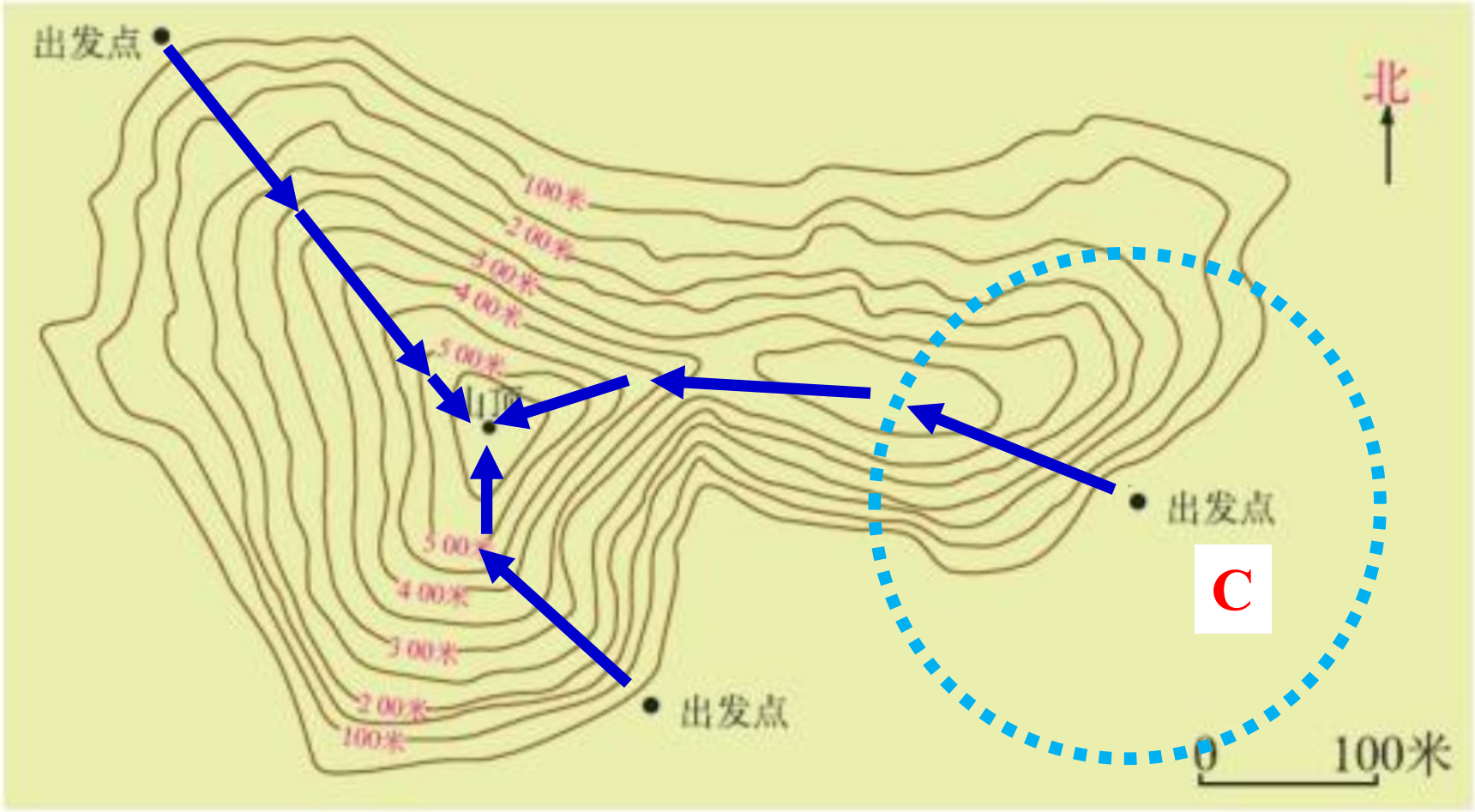
**Disadvantages:_____.**



# Large Neighborhood

**Advantages:** It is easy to find the global solution. It does not need many steps to find a good solution.

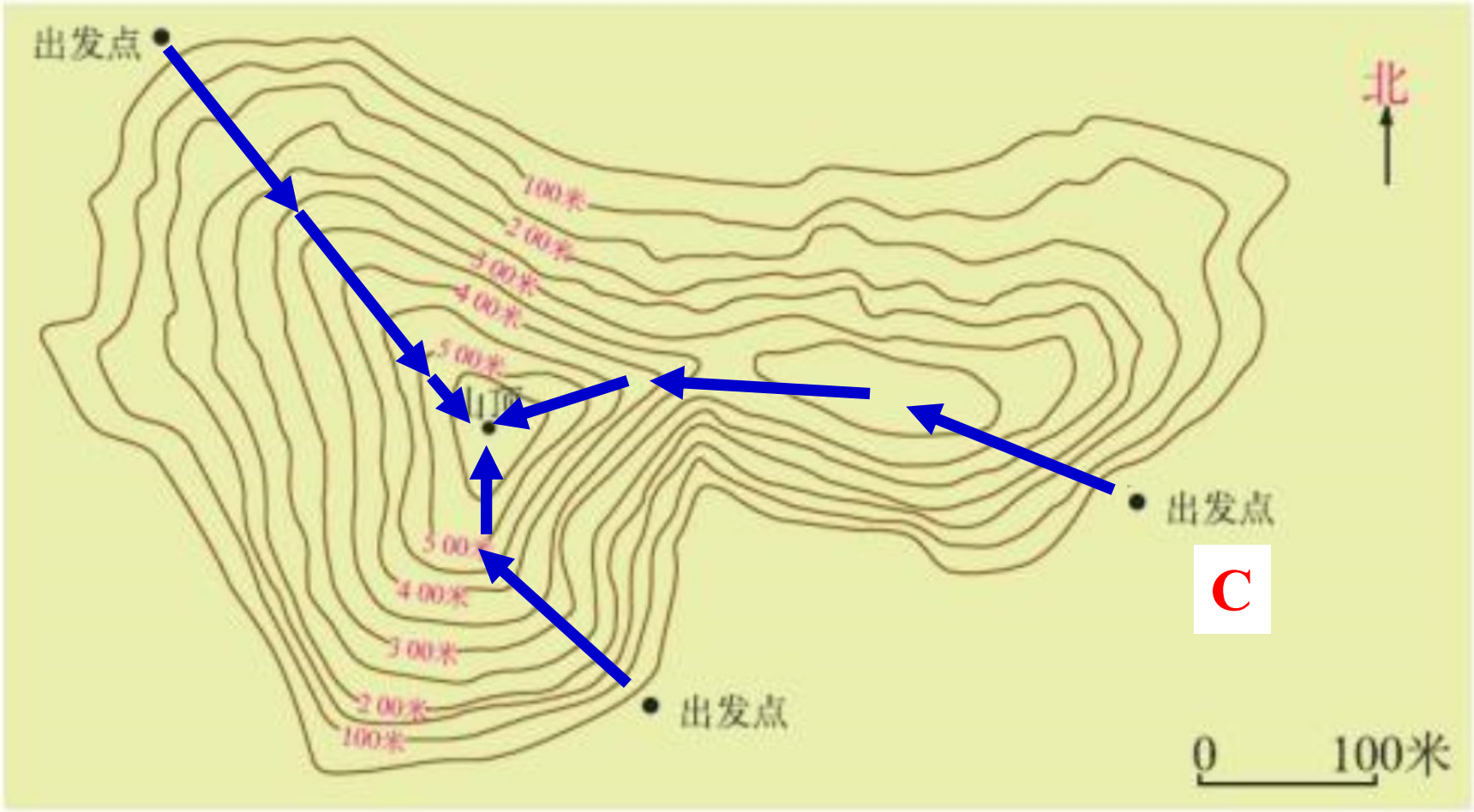
**Disadvantages:** Computation load is large for one step.



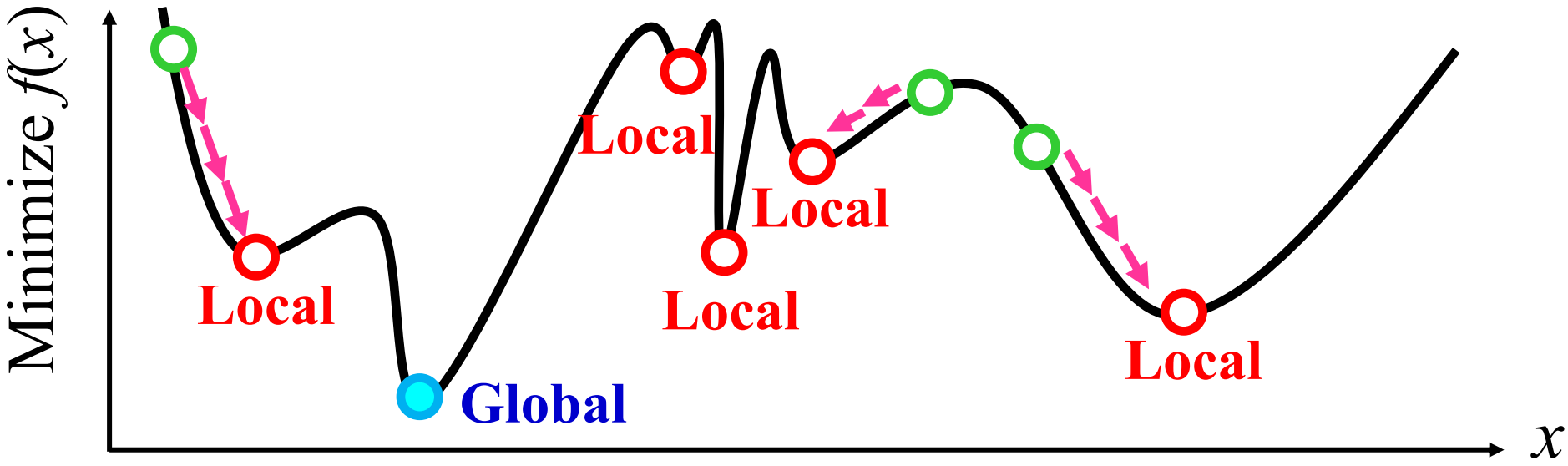
# Large Neighborhood looks better !

**Advantages:** It is easy to find the global solution. It does not need many steps to find a good solution.

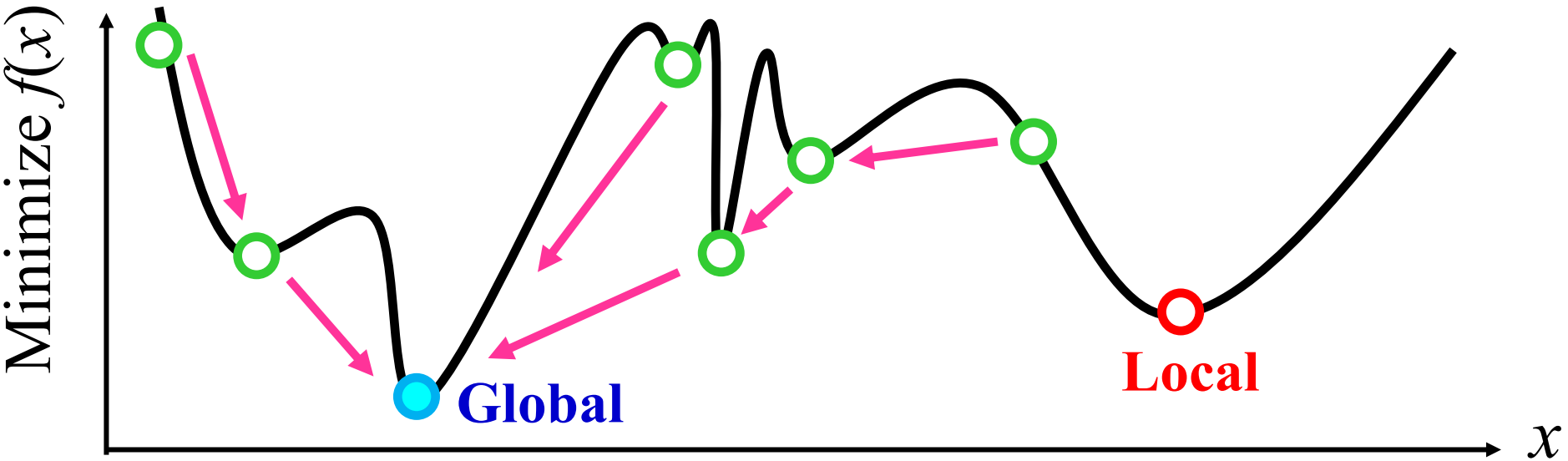
**Disadvantages:** Computation load is large for one step.

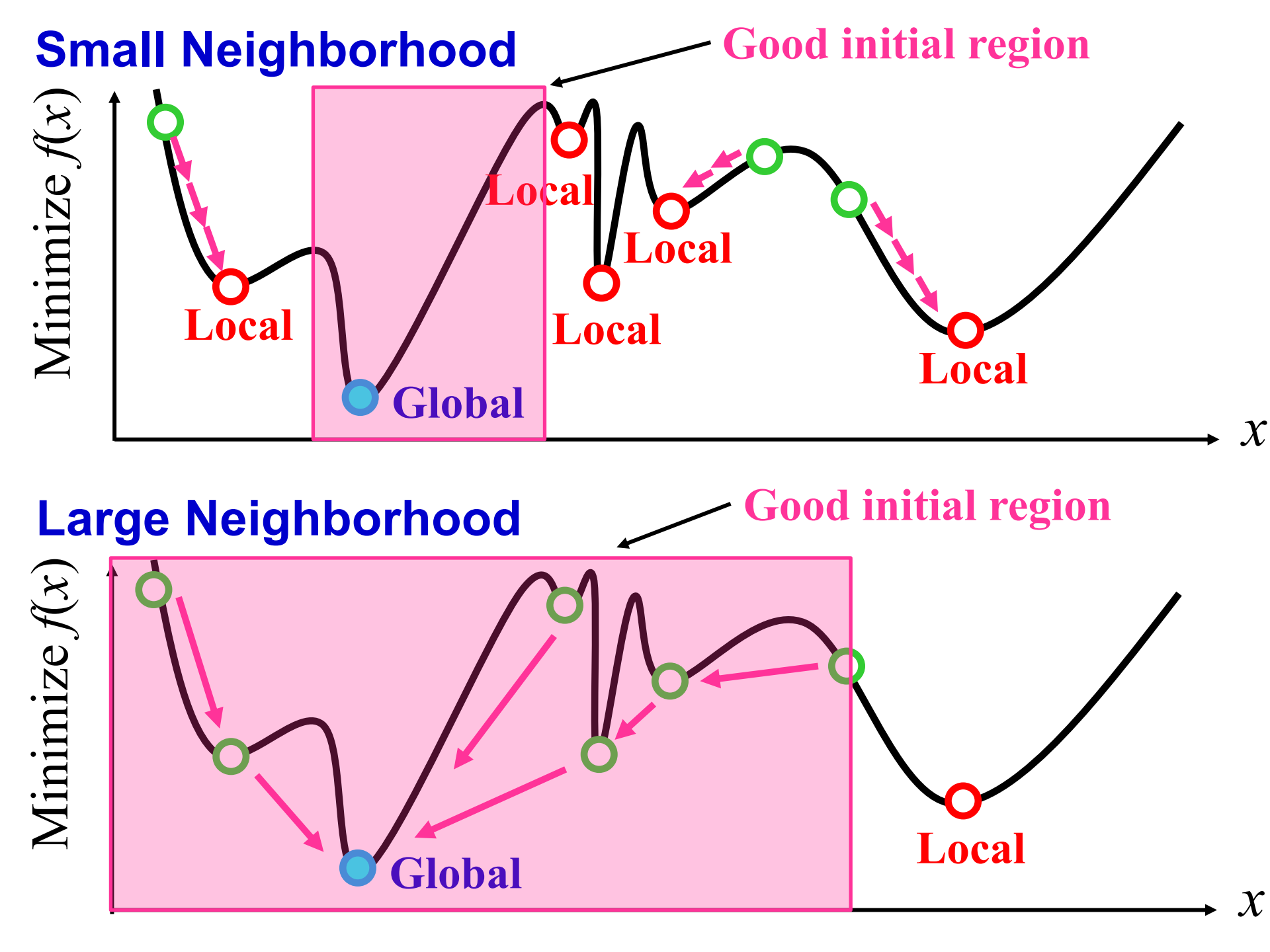


## Small Neighborhood




## Large Neighborhood




Small Neighborhood
Good initial region
Minimize f(x)
Local
Local
Local
Local
Local
Global
x
Large Neighborhood
Good initial region
Minimize f(x)
Global
Local
x

# Simple Counter-Example

Maximize $f(\boldsymbol{x}) = 2^1 x_1 + 2^2 x_2 + \ldots + 2^i x_i + \ldots + 2^{100} x_{100}$

subject to $x_i = 0$ or 1 ($i = 1, 2, \ldots, 100$)

**Setting of Experiments:**

Initial Solution: (0, 0, 0, ...., 0)

Optimal Solution: (1, 1, 1, ..., 1)

# Simple Counter-Example

Maximize $f(\boldsymbol{x}) = 2^1 x_1 + 2^2 x_2 + \ldots + 2^i x_i + \ldots + 2^{100} x_{100}$

subject to $x_i = 0$ or 1 ($i = 1, 2, \ldots, 100$)

**Setting of Experiments:**

Initial Solution: (0, 0, 0, ...., 0)

Optimal Solution: (1, 1, 1, ..., 1)

**Best Move-based Local Search**

(i) Find the best solution $\boldsymbol{y}$ in the neighborhood of the current solution $\boldsymbol{x}$.

(ii) If $\boldsymbol{y}$ is better than $\boldsymbol{x}$, replace the current solution $\boldsymbol{x}$ with $\boldsymbol{y}$. Otherwise terminate the execution of the algorithm.

# Simple Counter-Example

Maximize $f(\boldsymbol{x}) = 2^1 x_1 + 2^2 x_2 + \ldots + 2^i x_i + \ldots + 2^{100} x_{100}$

subject to $x_i = 0$ or 1 ($i = 1, 2, \ldots, 100$)

**Setting of Experiments:**

Initial Solution: (0, 0, 0, ...., 0)

Optimal Solution: (1, 1, 1, ..., 1)

**Best Move-based Local Search**

**Two Neighborhood Definitions:**

Small Neighborhood: Hamming Distance is 1.

Large Neighborhood: Hamming Distance is 1 or 2 (less than 3).

**Hamming Distance: The number of different elements**

Example: Hamming Distance(00000, 00101) = 2

## Two Neighborhood Definitions:

Small Neighborhood: Hamming Distance is 1.

Large Neighborhood: Hamming Distance is 1 or 2 (less than 3).

## Small Neighborhood

(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0): Current solution

(**1**, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0): Neighbor 1

(0, **1**, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0): Neighbor 2

(0, 0, **1**, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0): Neighbor 3

. . .

(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, **1**): Neighbor 100

## Two Neighborhood Definitions:

Small Neighborhood: Hamming Distance is 1.

Large Neighborhood: Hamming Distance is 1 or 2 (less than 3).

## Large Neighborhood

(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0): Current solution

(**1**, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0): Neighbor 1

(0, **1**, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0): Neighbor 2

(0, 0, **1**, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0): Neighbor 3

. . .

(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, **1**): Neighbor 100

(**1**, **1**, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0): Neighbor 101

(**1**, 0, **1**, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0): Neighbor 102

(**1**, 0, 0, **1**, 0, 0, 0, 0, 0, 0, ... 0, 0, 0): Neighbor 103

. . .

(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, **1**, **1**): Neighbor **????** **(to TA)**

## Two Neighborhood Definitions:

Small Neighborhood: Hamming Distance is 1.

Large Neighborhood: Hamming Distance is 1 or 2 (less than 3).

## Large Neighborhood

(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0): Current solution

(**1**, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0): Neighbor 1

(0, **1**, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0): Neighbor 2

(0, 0, **1**, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0): Neighbor 3

. . .

(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, **1**): Neighbor 100

(**1**, **1**, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0): Neighbor 101

(**1**, 0, **1**, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0): Neighbor 102

(**1**, 0, 0, **1**, 0, 0, 0, 0, 0, 0, ... 0, 0, 0): Neighbor 103

. . .

(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, **1**, **1**): Neighbor 5050

**Setting of Experiments:**

Initial Solution: (0, 0, 0, ...., 0)

Optimal Solution: (1, 1, 1, ..., 1)

Best Move-based Local Search

**Two Neighborhood Definitions:**

Small Neighborhood: Hamming Distance is 1.

Large Neighborhood: Hamming Distance is 1 or 2 (less than 3).

**The number of neighbors (Binary string of length 100):**

Hamming Distance is 1: ${}_{100}C_1 = 100$

Hamming Distance is 2: ${}_{100}C_2 = 100 \times 99/2 = 4950$

**Small Neighborhood: 100**

**Large Neighborhood: 100 + 4950 = 5050**

# Required Computation Load

**From (0, 0, ..., 0) to (1, 1, ..., 1) of string length 100**

**(1) Hamming Distance is 1: 100 x ??? evaluations (to TA)**

**Step 1:** Current solution is (0, 0, ..., 0). 100 neighbors such as (**1**, 0, ..., 0) and (0, **1**, 0, ..., 0) are examined, and (0, ..., 0, **1**) is chosen as the best neighbor.

Maximize $f(\boldsymbol{x}) = \mathbf{2^1}\, x_1 + \mathbf{2^2}\, x_2 + \ldots + \mathbf{2^i}\, x_i + \ldots + \mathbf{2^{100}}\, x_{100}$

subject to $x_i = 0$ or 1 ($i = 1, 2, \ldots, 100$)

# Required Computation Load

## From (0, 0, ..., 0) to (1, 1, ..., 1) of string length 100

### (1) Hamming Distance is 1: 100 x 101 evaluations

**Step 1:** Current solution is (0, 0, ..., 0). 100 neighbors such as (**1**, 0, ..., 0) and (0, **1**, 0, ..., 0) are examined, and (0, ..., 0, **1**) is chosen as the best neighbor.

**Step 2:** Current solution is (0, 0, 0, ..., 0, 1). 100 neighbors such as (**1**, 0, 0, ..., 0, 1) and (0, **1**, 0, ..., 0, 1) are examined, and (0, ..., 0, **1**, 1) is chosen as the best neighbor.

. . .

**Step 100:** Current solution is (0, 1, 1, ..., 1). 100 neighbors such as (**1**, 1, 1, ..., 1) and (0, **0**, 1, ..., 1) are examined, and (**1**, 1, 1, .., 1) is chosen as the best neighbor.

**Step 101**: Current solution is (1, 1, ..., 1). 100 neighbors such as (**0**, 1, ..., 1) and (1, **0**, 1, ..., 1) are examined. No better neighbor.

# Required Computation Load

**From (0, 0, ..., 0) to (1, 1, ..., 1) of string length 100**

**(2) Hamming Distance is 1 or 2: 5050 x ?? evaluations TA**

**Step 1:** Current solution is (0, 0, 0, ..., 0). 5050 neighbors such as (**1**, 0, 0, ..., 0) and (**1**, **1**, 0, ..., 0) are examined, and (0, ..., 0, **1**, **1**) is chosen as the best neighbor.

Maximize $f(\boldsymbol{x}) = \mathbf{2^1}\, x_1 + \mathbf{2^2}\, x_2 + \ldots + \mathbf{2^i}\, x_i + \ldots + \mathbf{2^{100}}\, x_{100}$

subject to $x_i = 0$ or 1 ($i = 1, 2, \ldots, 100$)

# Required Computation Load

## From (0, 0, ..., 0) to (1, 1, ..., 1) of string length 100

## (2) Hamming Distance is 1 or 2: 5050 x 51 evaluations

**Step 1:** Current solution is (0, 0, 0, ..., 0). 5050 neighbors such as (**1**, 0, 0, ..., 0) and (**1**, **1**, 0, ..., 0) are examined, and (0, ..., 0, **1**, **1**) is chosen as the best neighbor.

**Step 2:** Current solution is (0, 0, 0, ..., 0, 1, 1). 5050 neighbors such as (**1**, 0, 0, ..., 0, 1, 1) and (**1**, **1**, 0, ..., 0, 1, 1) are examined, and (0, ..., 0, **1**, **1**, 1, 1) is chosen as the best neighbor.

. . .

**Step 50:** Current solution is (0, 0, 1, ..., 1). 5050 neighbors such as (**1**, 0, 1, ..., 1) and (**1**, **1**, 1, ..., 1) are examined, and (**1**, **1**, 1, .., 1) is chosen as the best neighbor.

**Step 51**: Current solution is (1, 1, ..., 1). 5050 neighbors such as (**0**, 1, ..., 1) and (**0**, **0**, 1, ..., 1) are examined. No better neighbor.

# Required Computation Load

**From (0, 0, ..., 0) to (1, 1, ..., 1) of string length 100**

**(1) Hamming Distance is 1: 10,100 evaluations**

- The number of neighbors to be examined: 100
- The step size of local search: One bit
- The number of steps to the optimal solution: 100 steps
- The number of examined solutions: 100x100 + 100

(+ 100: for termination)

**(2) Hamming Distance is 1 or 2: 257,550 evaluations**

- The number of neighbors to be examined: 5050
- The step size of local search: Two bits
- The number of steps to the optimal solution: 50 steps
- The number of examined solutions: 5050x50 + 5050

(+ 5050: for termination)

# Required Computation Load

**From (0, 0, ..., 0) to (1, 1, ..., 1) of string length 100**

**(1) Hamming Distance is 1: 10,100 evaluations (3.92%)**

- The number of neighbors to be examined: 100
- The step size of local search: One bit
- The number of steps to the optimal solution: 100 steps
- The number of examined solutions: 100x100 + 100

(+ 100: for termination)

**(2) Hamming Distance is 1 or 2: 257,550 evaluations**

- The number of neighbors to be examined: 5050
- The step size of local search: Two bits
- The number of steps to the optimal solution: 50 steps
- The number of examined solutions: 5050x50 + 5050

(+ 5050: for termination)

# First Move Strategy

(i) Generate a neighbor solution $\boldsymbol{y}$ of the current solution $\boldsymbol{x}$.
(ii) If $\boldsymbol{y}$ is better than $\boldsymbol{x}$, replace the current solution $\boldsymbol{x}$ with $\boldsymbol{y}$.
(iii) If no better solution exists in the neighborhood of $\boldsymbol{x}$,
terminate the execution of the algorithm.

## Discussions:

Please consider whether we can obtain a similar conclusion for the case of the first move-based local search (i.e., the smaller neighborhood is faster than the larger neighborhood)?

# Large Neighborhood in First Move-Based Local Search

In the early stage of search (i.e., when almost all $x_i$ are zero), the large neighborhood may be faster since a single solution examination may lead to a two-bit improvement.

00000 00000 ... 00000 ==> 0001**1**0 00000 ... 0**1**000

00010 00000 ... 01000 ==> 00010 00**1**00 ... 0101**1**0

# Large Neighborhood in First Move-Based Local Search

In the early stage of search (i.e., when almost all $x_i$ are zero), the large neighborhood may be faster since a single solution examination may lead to a two-bit improvement.

00000 00000 ... 00000 ==> 000**1**0 00000 ... 0**1**000

00010 00000 ... 01000 ==> 00010 00**1**00 ... 010**1**0

However, in the final stage of search (i.e., when almost all $x_i$ are one), the larger neighborhood may be slower since many solution examinations may lead to a single-bit or zero-bit improvement.

10111 11111 ... 11011 ==> 101**0**1 11**0**11 ... 11011 (NG)

10111 11111 ... 11011 ==> 10111 11111 ... 11**1**0**1** (NG)

10111 11111 ... 11011 ==> 10111 11**0**11 ... 11**1**11 (Good!)

10111 11011 ... 11111 ==> 10111 **0**1011 ... 1**0**111 (NG)

10111 11011 ... 11111 ==> 1**1**111 1101**0** ... 11111 (NG)

# Discussions:

Please consider whether we can obtain a similar conclusion for the case of the first move-based local search (i.e., the smaller neighborhood is faster than the larger neighborhood)?

It seems that the large neighborhood is slower than the small neighborhood independent of the choice between the best move-based local search and the first move-based local search.

# Discussions:

Please consider whether we can obtain a similar conclusion for the case of the first move-based local search (i.e., the smaller neighborhood is faster than the larger neighborhood)?

# Extreme Case Analysis

The largest neighborhood is the same as the feasible region (i.e., the set of all solutions). In this case, the best move-based local search is the same as the examination of all solutions, and the first move-based local search is the same as random search. Both are not efficient.

# Discussions:

Please consider whether we can obtain a similar conclusion for the case of the first move-based local search (i.e., the smaller neighborhood is faster than the larger neighborhood)?

# Extreme Case Analysis

The largest neighborhood is the same as the feasible region (i.e., the set of all solutions). In this case, the best move-based local search is the same as the examination of all solutions, and the first move-based local search is the same as random search. Both are not efficient.

*** It is useful for understanding algorithm behavior to examine some extreme cases (e.g., the smallest neighborhood, the largest neighborhood, the worst starting point).**

# Local Search for Knapsack Problem

$$\text{Maximize} \quad f(\boldsymbol{x}) = \sum_{i=1}^{n} v_i x_i$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \le W \quad \text{and} \quad x_i \in \{0, 1\},\ i = 1, 2, \ldots, n$$

Solution representation (binary string): $\boldsymbol{x} = x_1 x_2 \ldots x_n$
Initial solution of local search: Greedy solution
**Select items in a decreasing order of value/weight ($v_i/w_i$).**
[Point] Examine all items without stopping the algorithm.

Initial Solution (Greedy Solution): 1011010100000011
(i) By changing "1" to "0": $f(\boldsymbol{x})$ is always decreased.
(ii) By changing "0" to "1": Constraint is not satisfied.
We cannot improve the greedy solution by local search if the neighborhood structure is defined by "Hamming Distance is 1"

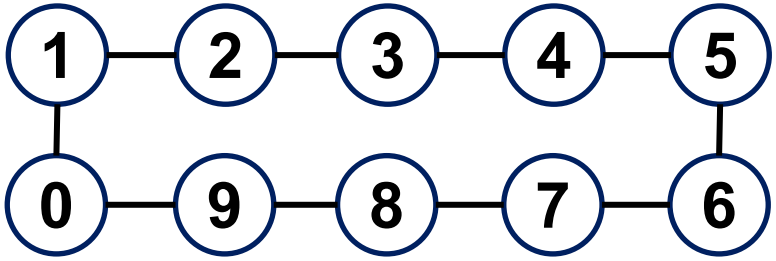
# Local Search for TSP

## How to specify the neighborhood of the current solution ?

**Current solution:** Permutation of $n$ cities.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|

**Corresponding tour:**



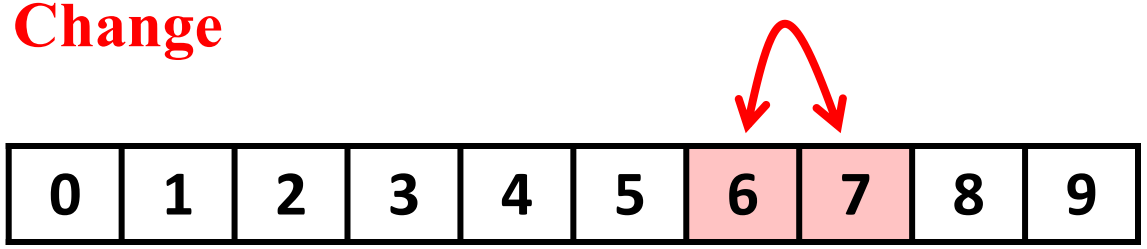

**Neighborhood:**

A set of solutions which are generated by slightly changing the current solution. (Neighborhood Structure: Mechanism to generate neighboring solutions).

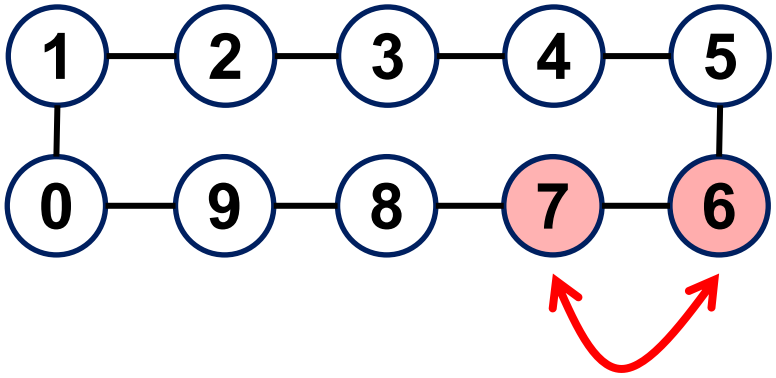
# Lab Session Task:

Show the number of neighbors created by each of the following neighborhood structures for an n-city TSP problem.

## (1) Adjacent Two-City Change

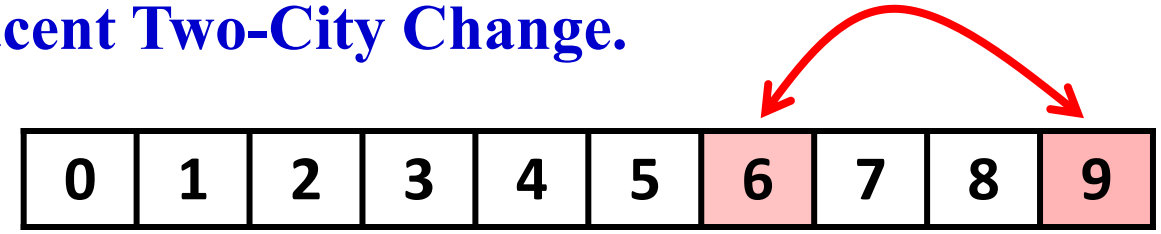



Number of neighbors: ?

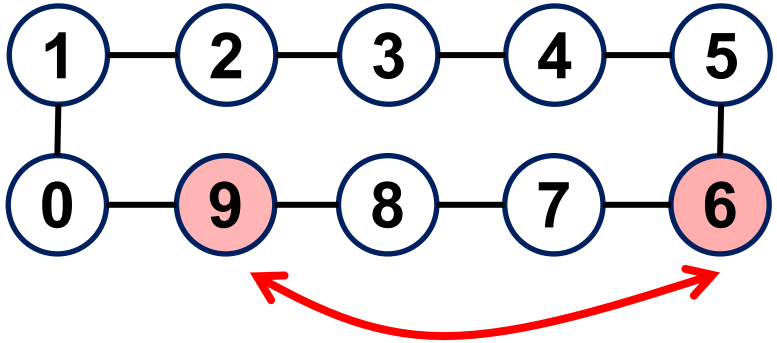


## (2) Arbitrary Two-City Change

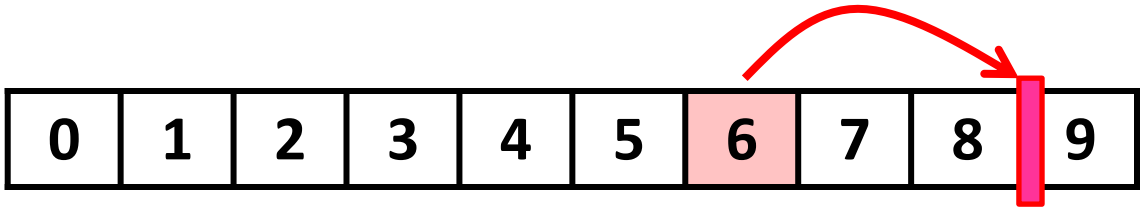
**including (1) Adjacent Two-City Change.**



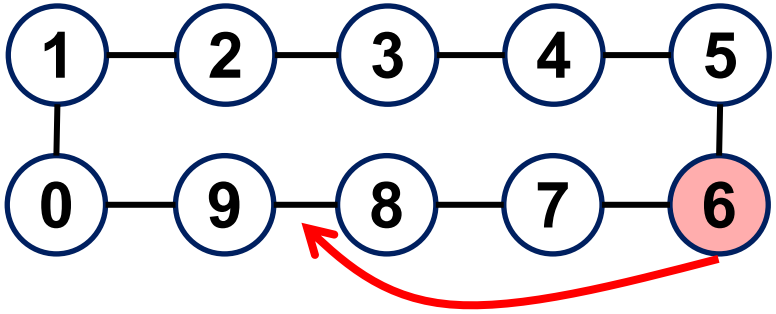

Number of neighbors: ?



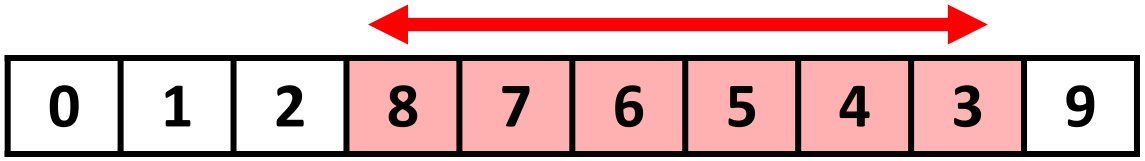

## (3) Insertion (Shift)



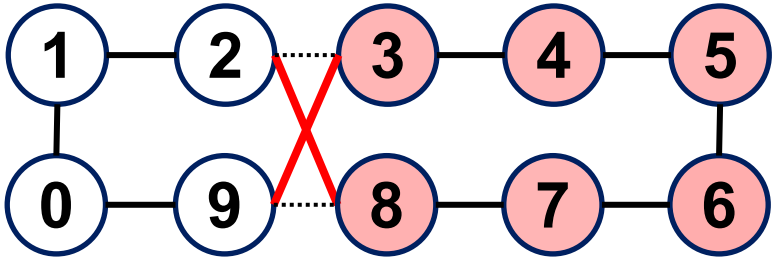

Number of neighbors: ?



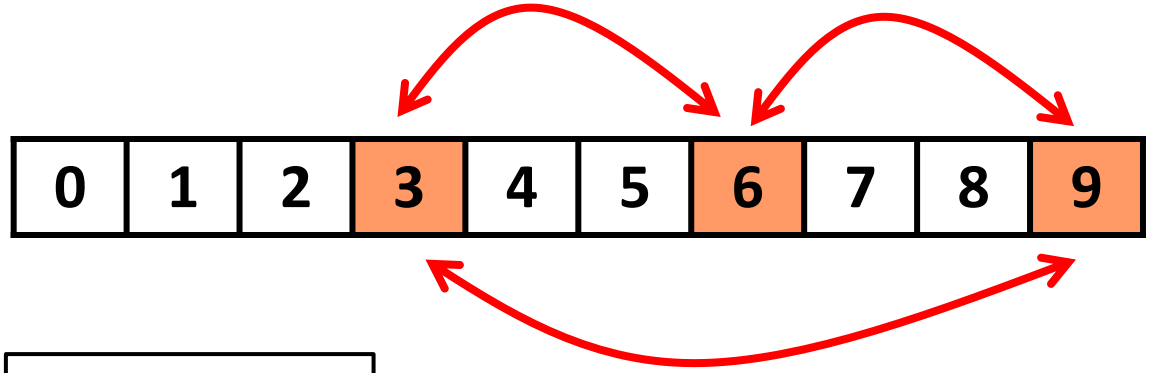
## (4) Inversion (Arbitrary Two-Edge Change)




Number of neighbors: ?




## (5) Arbitrary Three-City Change
**including two-city change in (1) and (2).**




Number of neighbors: ?