Optimization and Simulation

Optimization

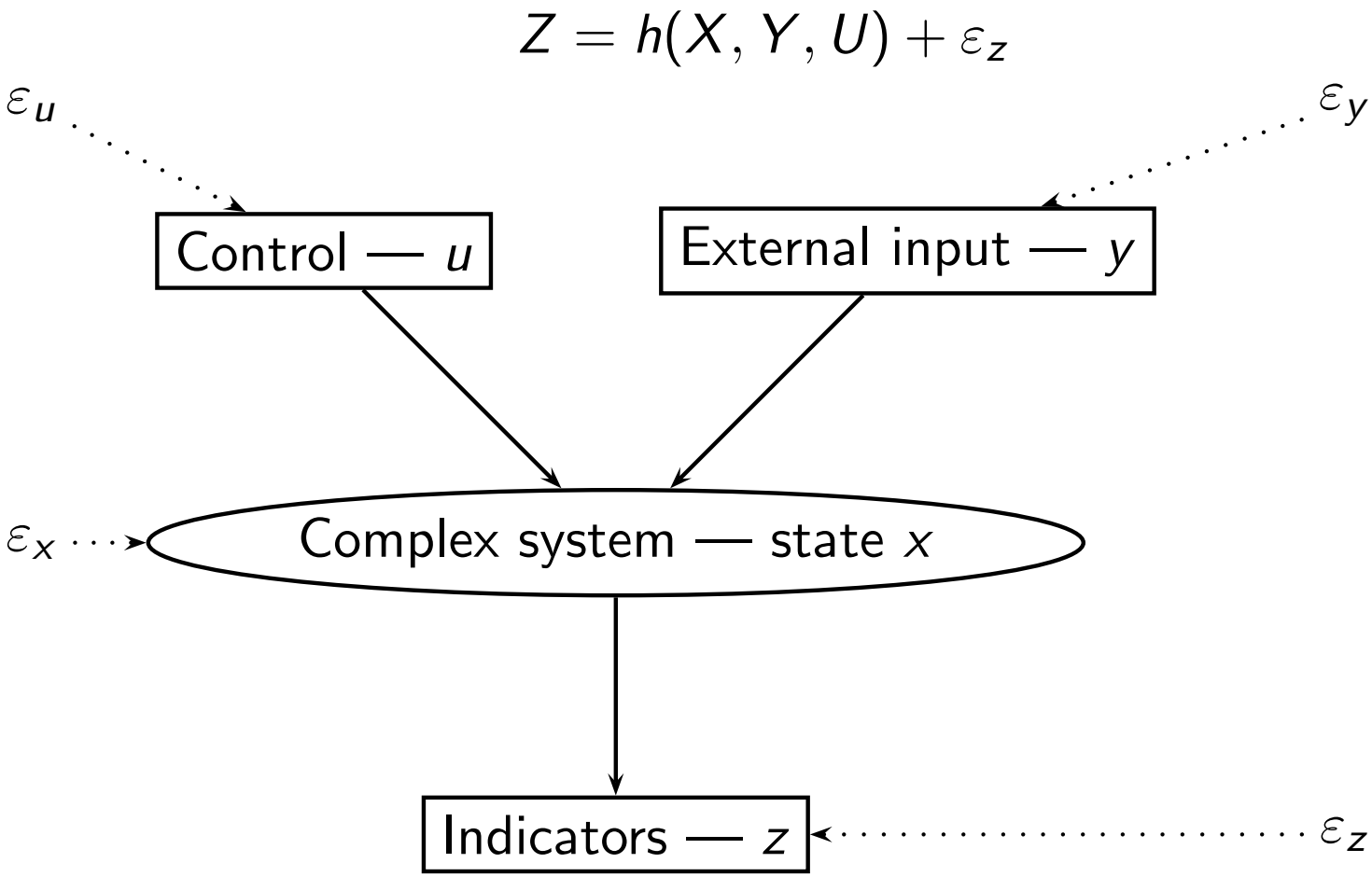
Michel Bierlaire

Transport and Mobility Laboratory
School of Architecture, Civil and Environmental Engineering
Ecole Polytechnique Fédérale de Lausanne

EPFL

Outline

General framework



General framework

Assumptions

- Control U is deterministic.

$$Z(u) = h(X, Y, u) + \varepsilon_z$$

- Various features of Z are considered: mean, variance, quantile, etc.

$$(z_1(u), \ldots, z_m(u))$$

- They are combined in a single indicator:

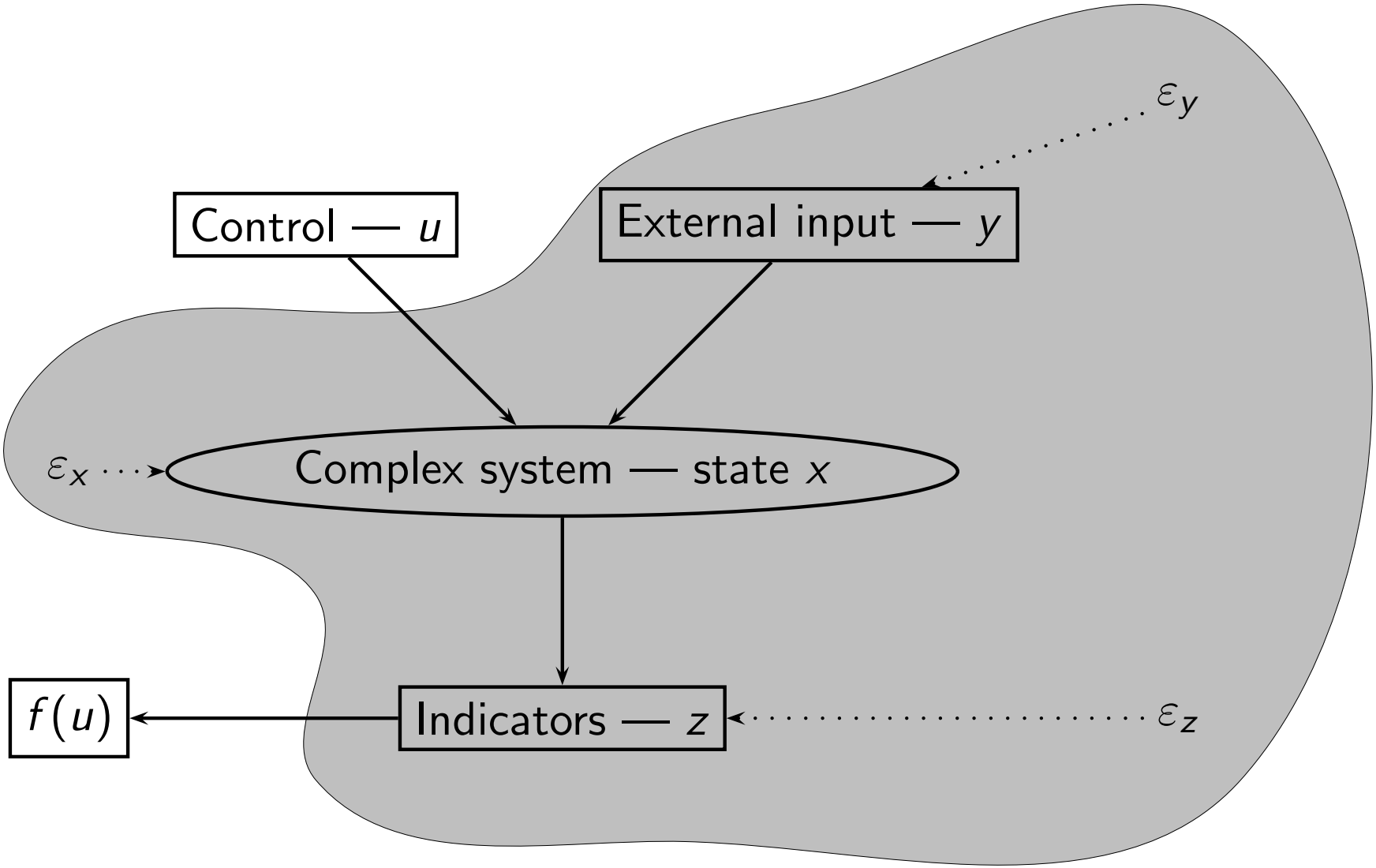
$$f(u) = g(z_1(u), \ldots, z_m(u))$$

General framework: example

Rico at Satellite

- X: number of customers in the bar
- Y: arrivals of customers
- u: service time of Rico
- $Z(u)$: waiting time of the customers
- $z_1(u)$: mean waiting time
- $z_2(u)$: maximum waiting time
- $f(u) = g(z_1(u), z_2(u)) = z_1 + z_2$

General framework: the black box



Optimization problem

$$\min_{u \in \mathbb{R}^n} f(u)$$

subject to

$$u \in \mathcal{U} \subseteq \mathbb{R}^n$$

- u: decision variables
- $f(u)$: objective function
- $u \in \mathcal{U}$: constraints
- $\mathcal{U}$: feasible set

If $\mathcal{U}$ is a finite set: **combinatorial optimization**.

Outline

The knapsack problem

- Patricia prepares a hike in the mountain.
- She has a knapsack with capacity Wkg.
- She considers carrying a list of n items.
- Each item has a utility u_i and a weight w_i.
- What items should she take to maximize the total utility, while fitting in the knapsack?

Modeling

Decision variables

$$x_i = \begin{cases} 1 & \text{if item } i \text{ goes into the knapsack,} \\ 0 & \text{otherwise} \end{cases}$$

Objective function

$$\max f(x) = \sum_{i=1}^{n} u_i x_i$$

Constraints

$$\sum_{i=1}^{n} w_i x_i \leq W$$

$$x_i \in \{0,1\} \quad i = 1, \ldots, n$$

Brute force algorithm

Enumeration

- As $\mathcal{U}$ is finite, all solutions can be enumerated.
- Each object can be in or out, for a total of 2^n combinations.
- For each of them, we must:
  - Check that the weight is feasible.
  - If so, calculate the utility and check if it is the largest so far.

Computational time

- About $2n$ floating point operations per combination.
- Assume a 1 Teraflops processor: 10^{12} floating point operations per second.

Brute force algorithm

Computational time

- If $n = 34$, about 1 second to solve.
- If $n = 40$, about 1 minute.
- If $n = 45$, about 1 hour.
- If $n = 50$, about 1 day.
- If $n = 58$, about 1 year.
- If $n = 69$, about 2583 years, more than the Christian Era.
- If $n = 78$, about 1,500,000 years, time elapsed since Homo Erectus appeared on earth.
- If $n = 91$, about 10^{10} years, roughly the age of the universe.

Combinatorial optimization

- The set of feasible solutions is finite.
- But the number of feasible solutions grows exponentially with the size of the problem.
- No optimality condition can be exploited.

Traveling salesman problem

The problem

- Consider n cities.
- For any pair (i, j) of cities, the distance d_{ij} between them is known.
- Find the shortest possible itinerary that starts from the home town of the salesman, visit all other cities, and come back to the origin.

Feasible solutions

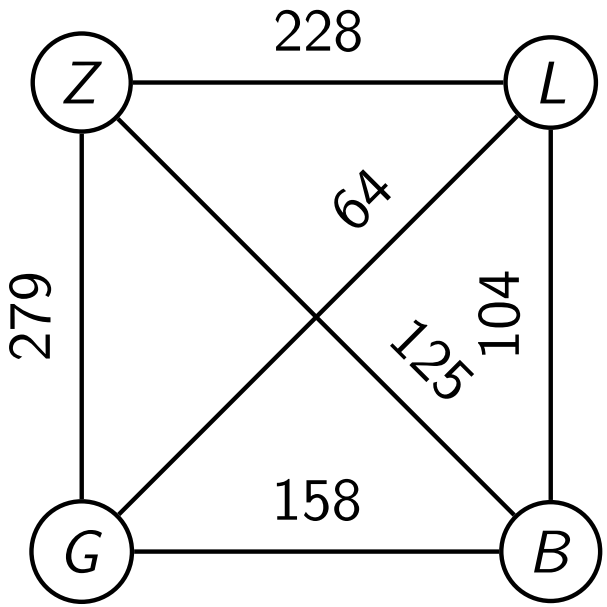
- Number of feasible solutions:

$$(n-1)! \approx \sqrt{2\pi/n-1)} \left(\frac{n-1}{e}\right)^{n-1}$$

- Again, the number of feasible solutions grows exponentially with the size of the problem.

TSP: example

Lausanne, Geneva, Zurich, Bern



Home town: Lausanne

3 possibilities:

- L $\rightarrow$ B $\rightarrow$ Z $\rightarrow$ G $\rightarrow$ L: 572 km
- L $\rightarrow$ B $\rightarrow$ G $\rightarrow$ Z $\rightarrow$ L: 769 km
- L $\rightarrow$ Z $\rightarrow$ B $\rightarrow$ G $\rightarrow$ L: 575 km

Integer linear optimization problem

Linear optimization

$$\min_{x \in \mathbb{R}^n} c^T x$$

subject to

$$Ax = b$$
$$x \geq 0.$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$.

Integer Linear optimization

$$\min_{x \in \mathbb{R}^n} c^T x$$

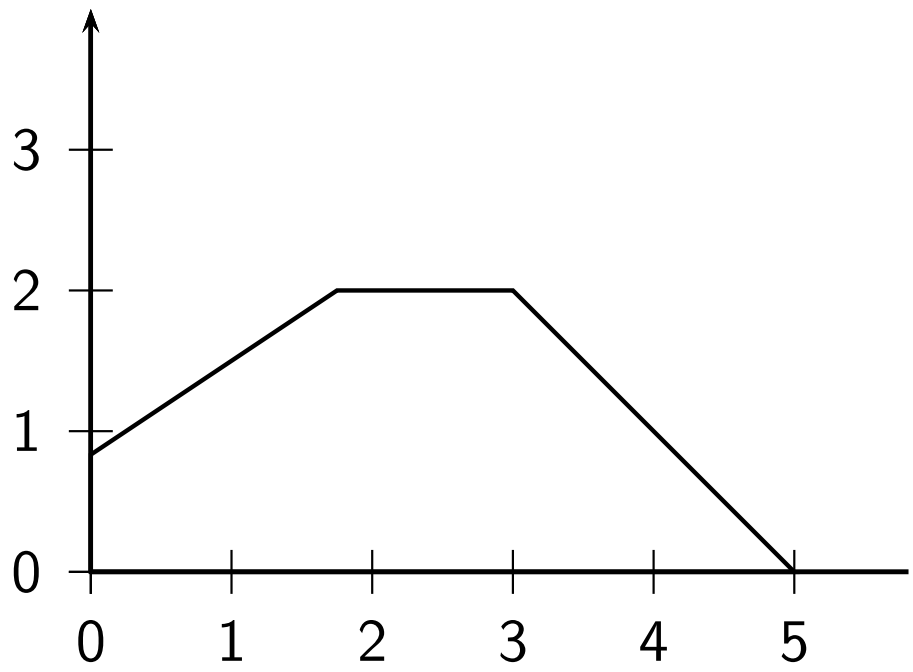
subject to

$$Ax = b$$
$$x \in \mathbb{N}.$$

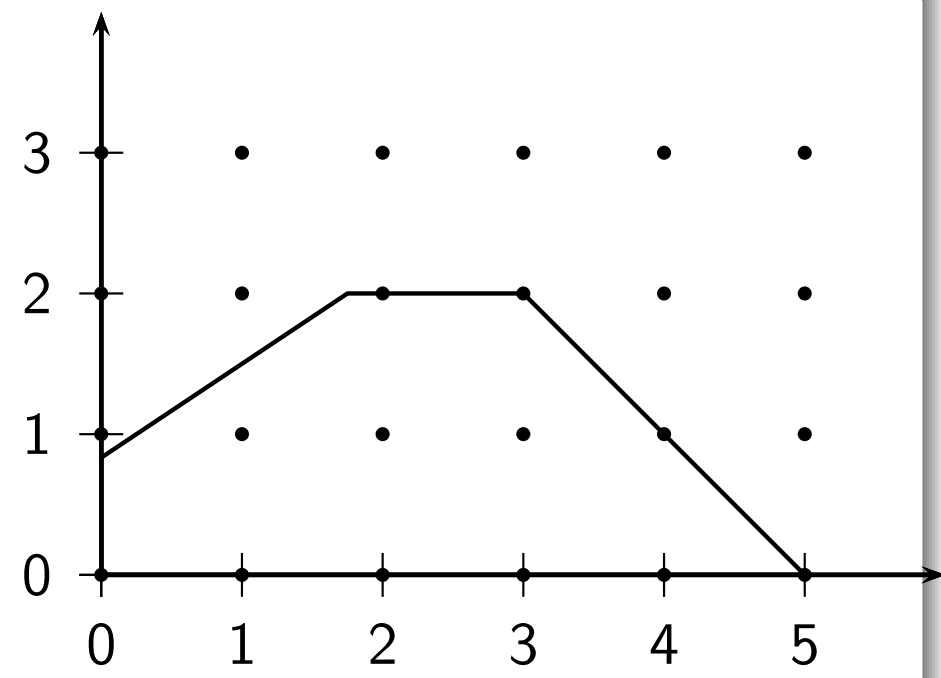
where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$.

Feasible set

Polyhedron



Intersection polyhedron/integer lattice



Outline

Greedy heuristics

Principles

- Step by step construction of a feasible solution.
- At each step, a local optimization is performed.
- Decisions taken at previous steps are definitive.

Properties

- Easy to implement.
- Short computational time.
- May generate poor solutions.
- Used to generate initial solutions.

Greedy heuristics

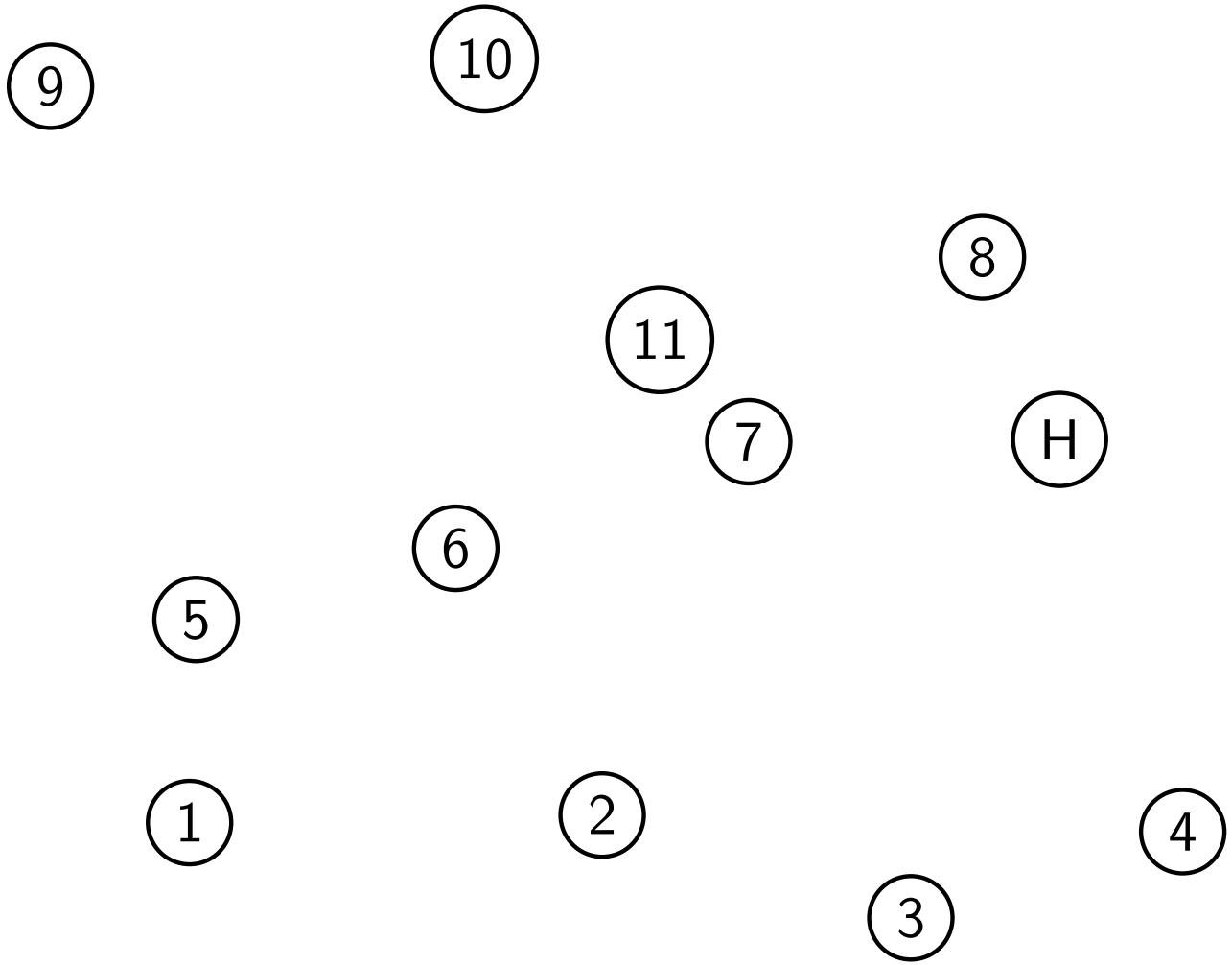
The knapsack problem

- Sort the items by decreasing order of u_i/w_i.
- For each item in this order, put it in the sack if it fits.

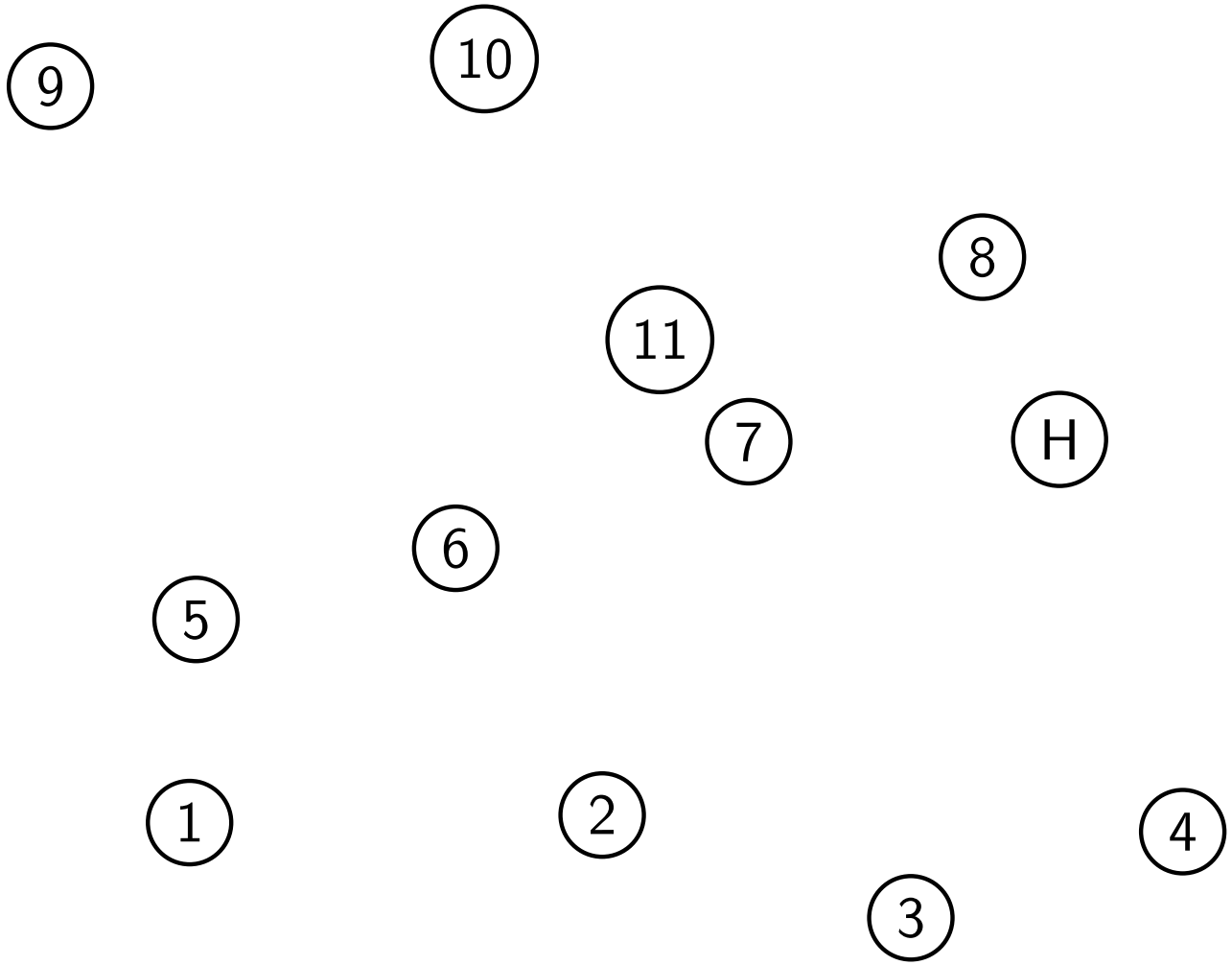
The traveling salesman problem

- Start from home.
- At each step, select the closest city as the next one.

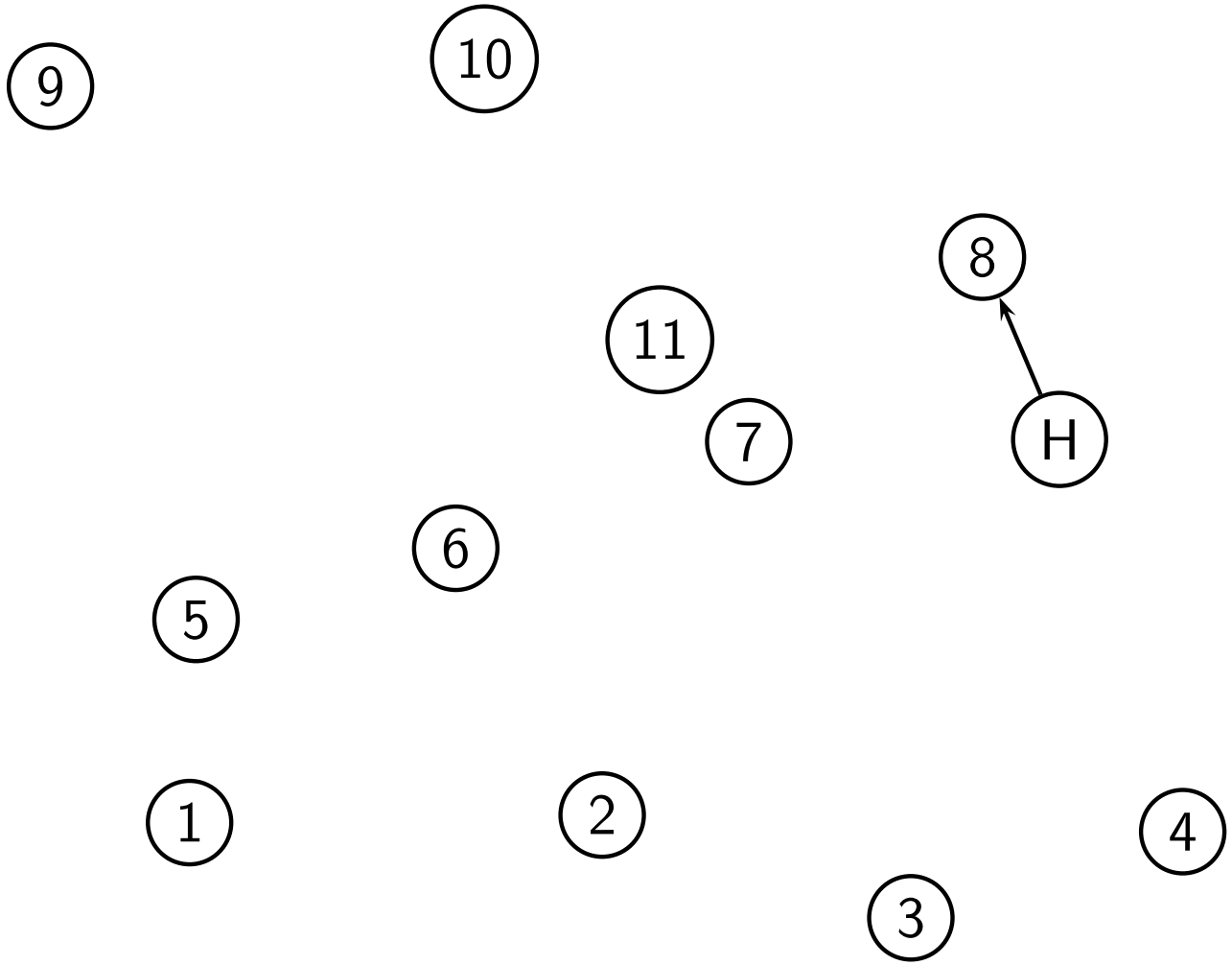
TSP: 12 cities (euclidean dist.)



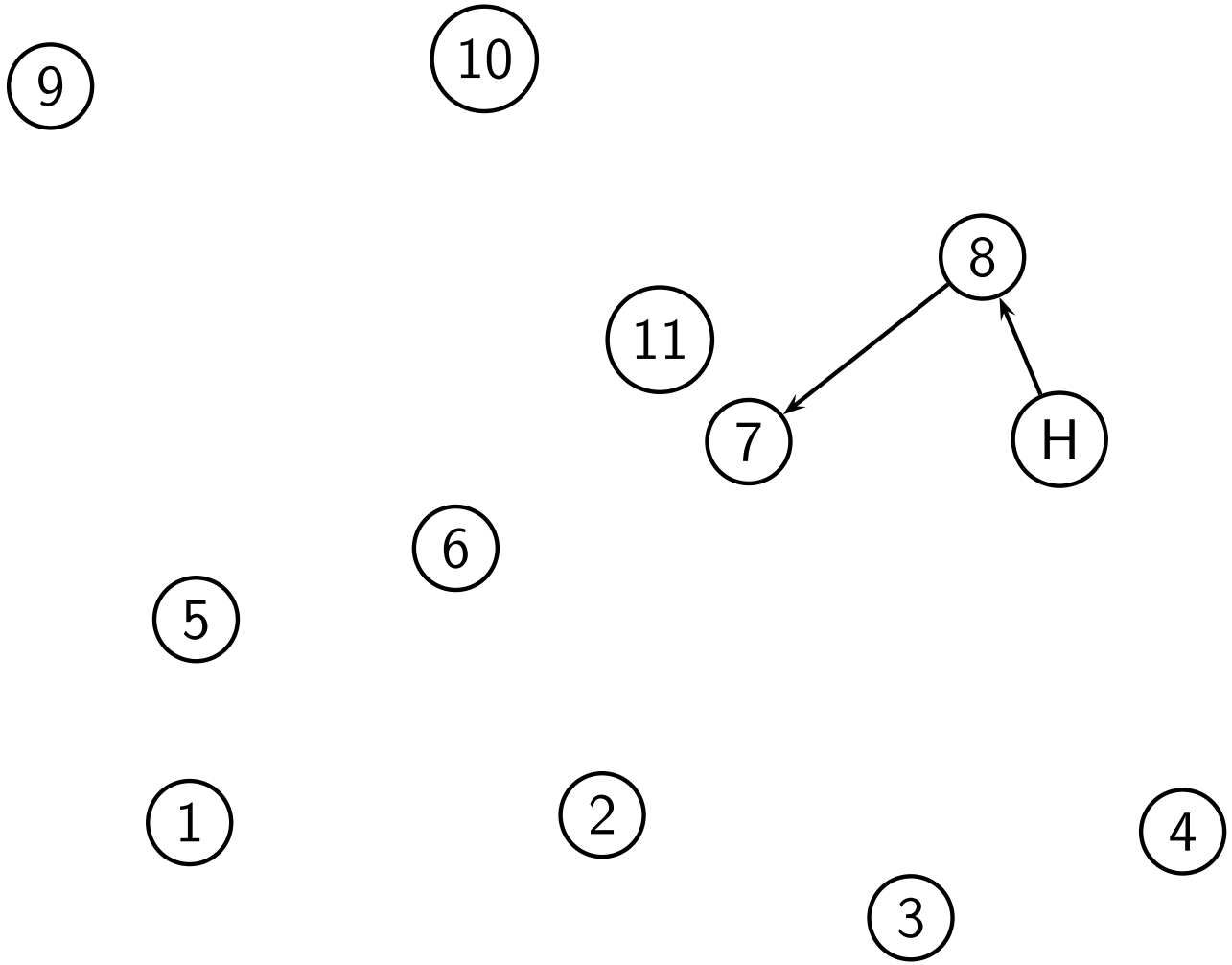
TSP: 12 cities



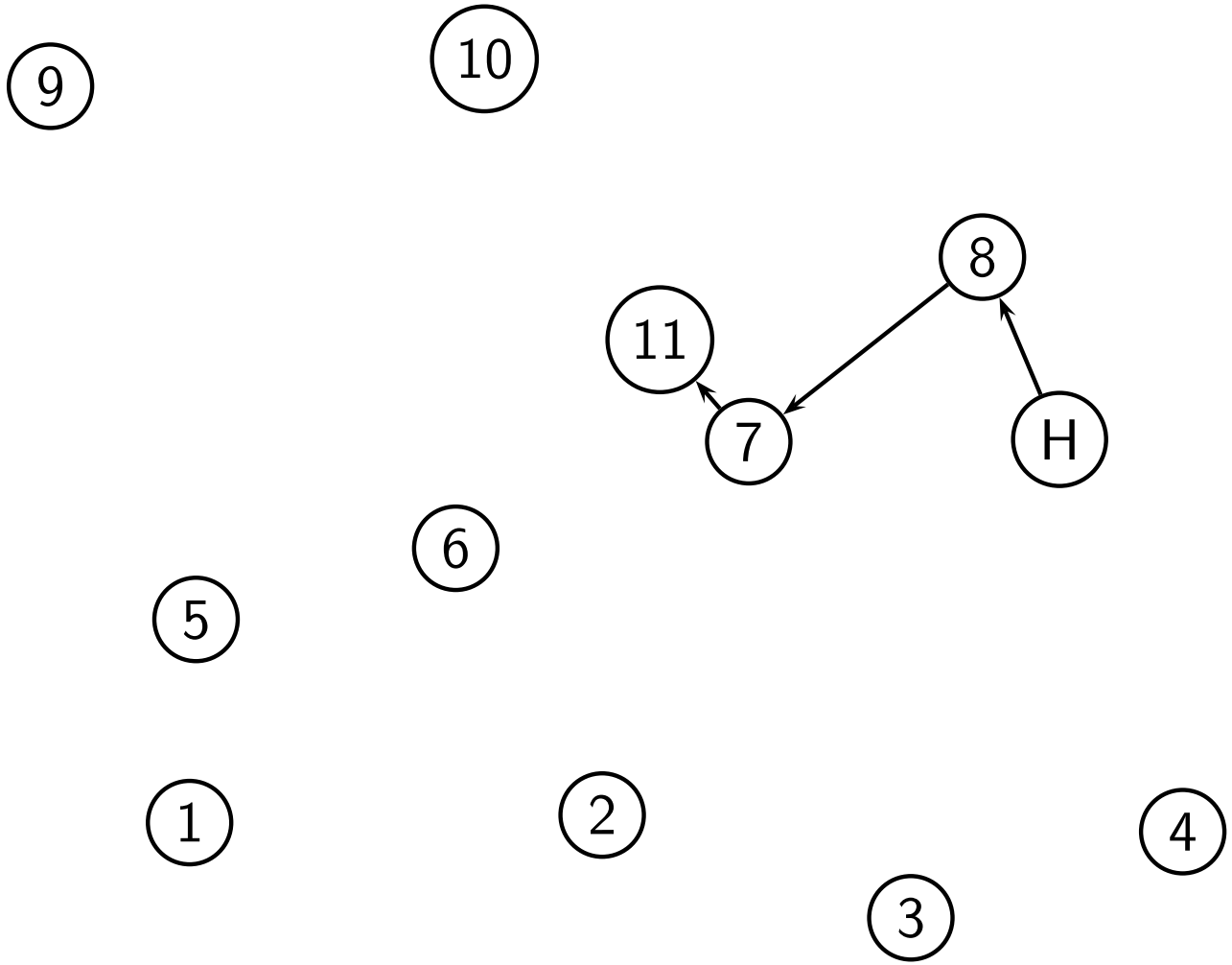
TSP: 12 cities



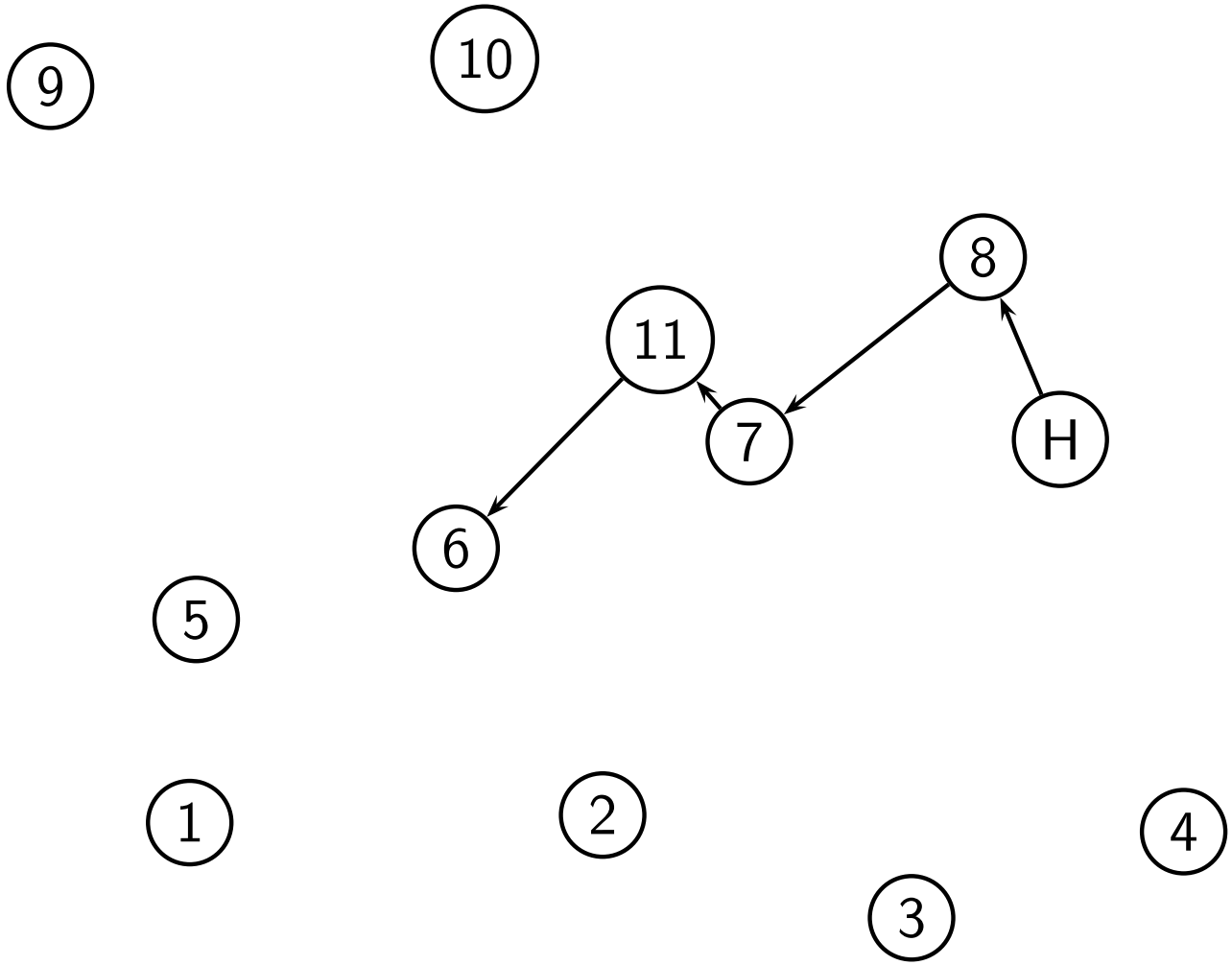
TSP: 12 cities



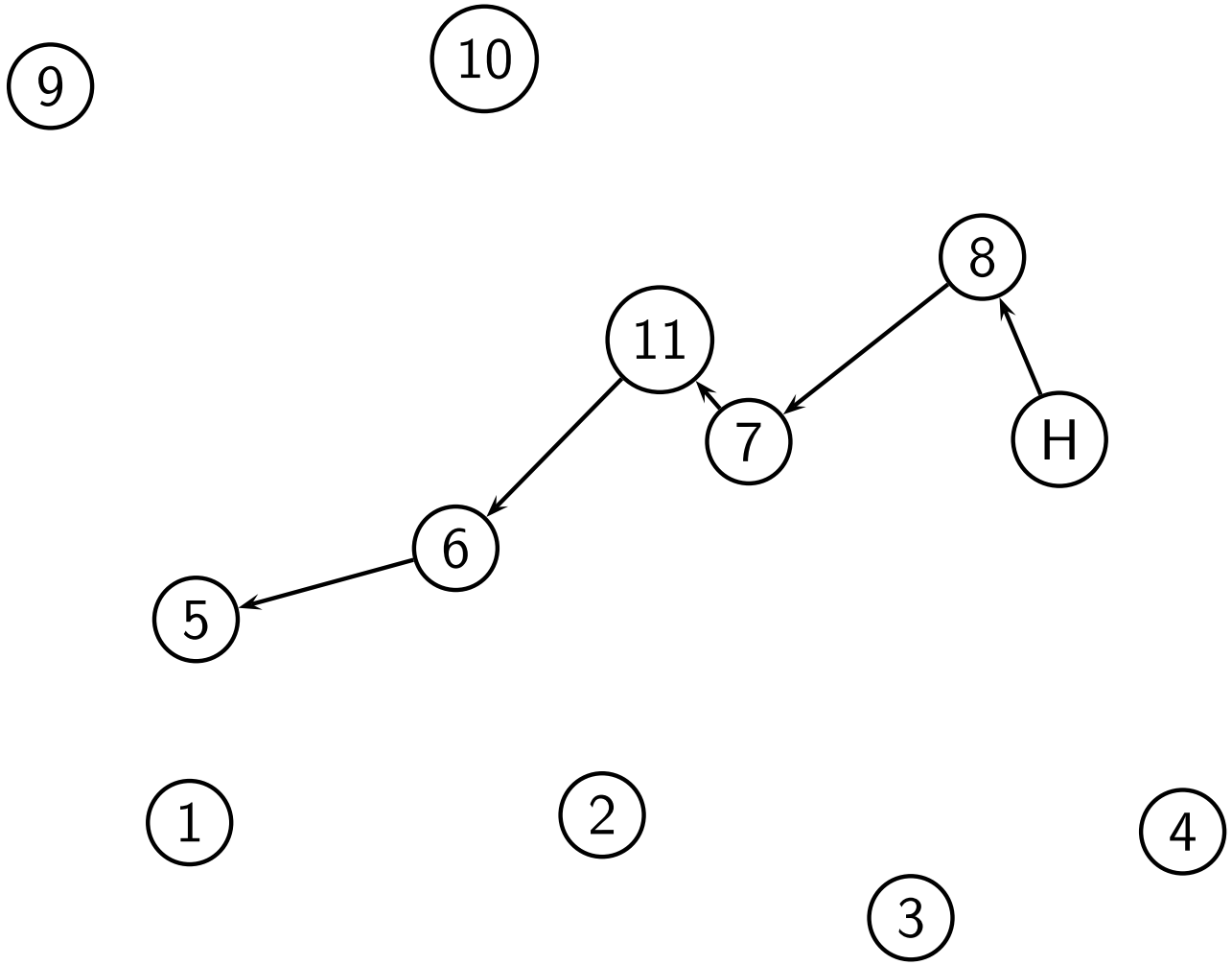
TSP: 12 cities



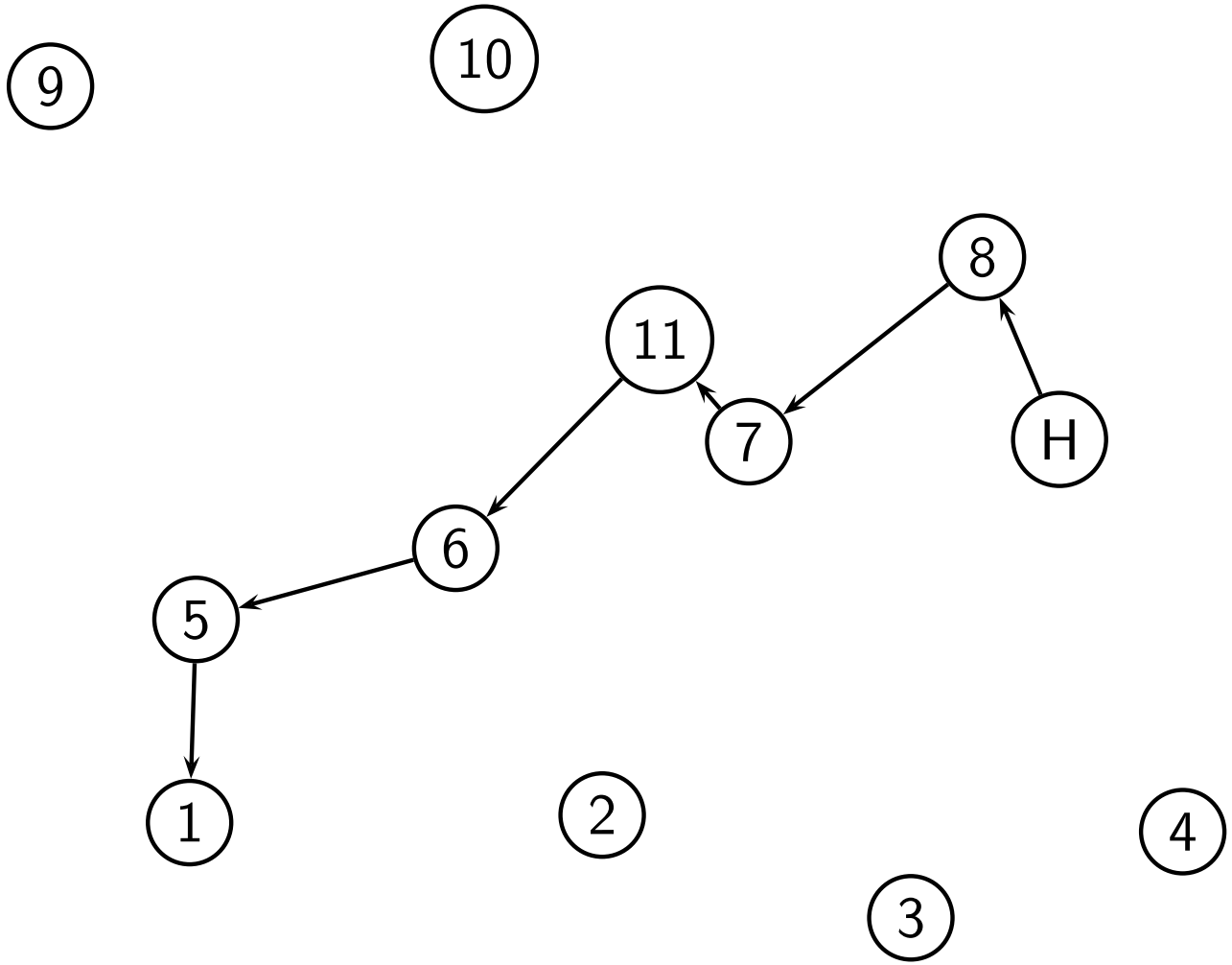
TSP: 12 cities



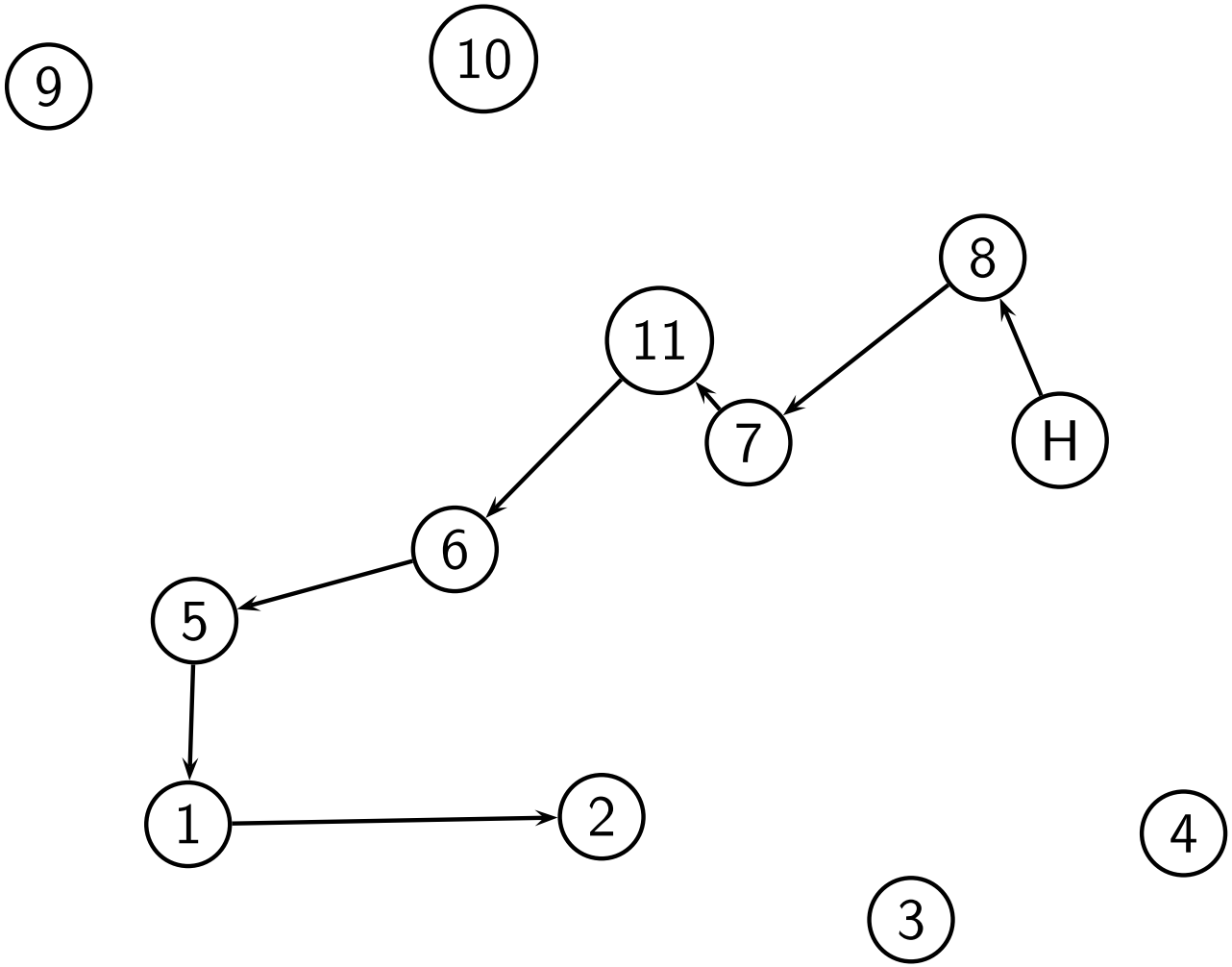
TSP: 12 cities



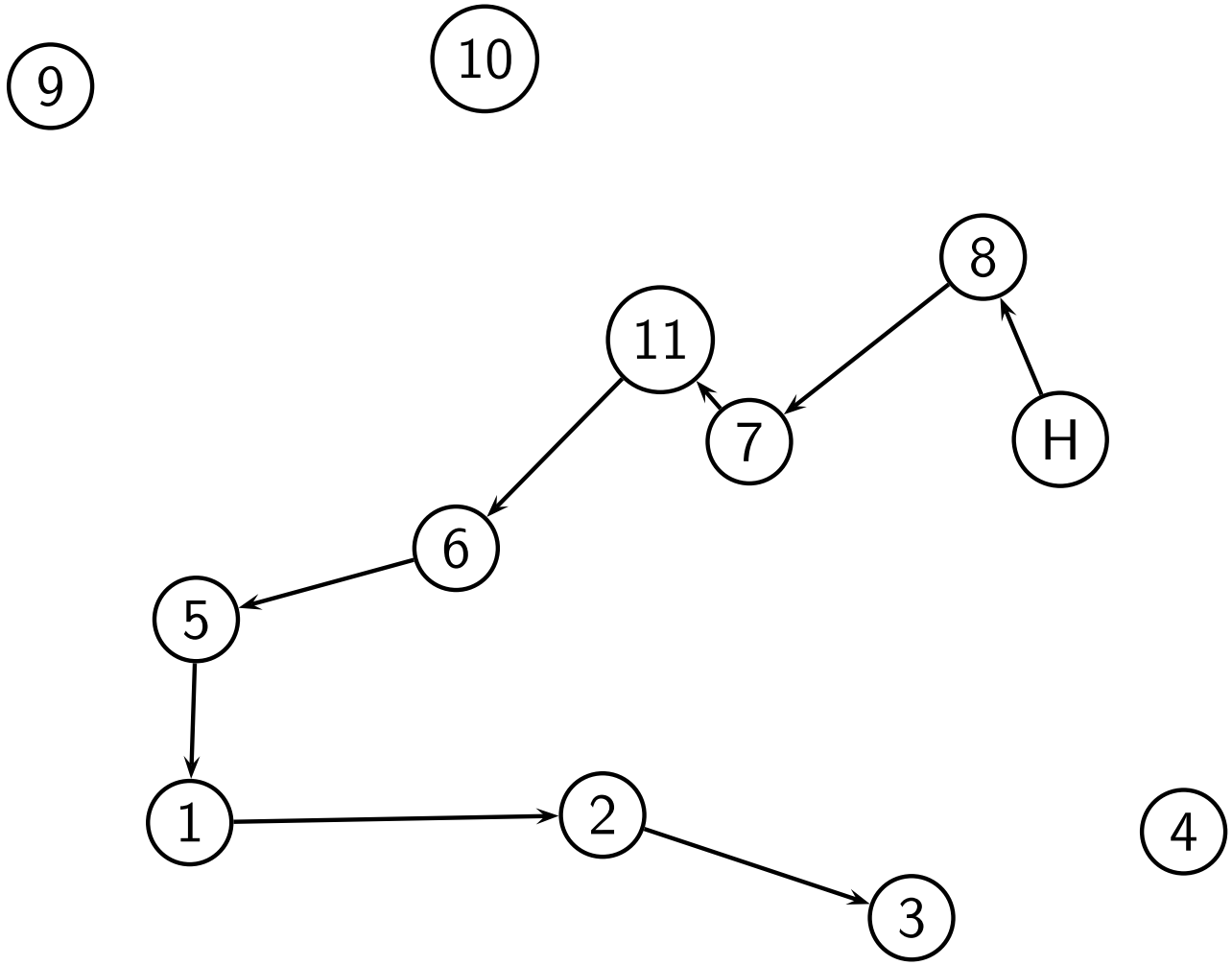
TSP: 12 cities



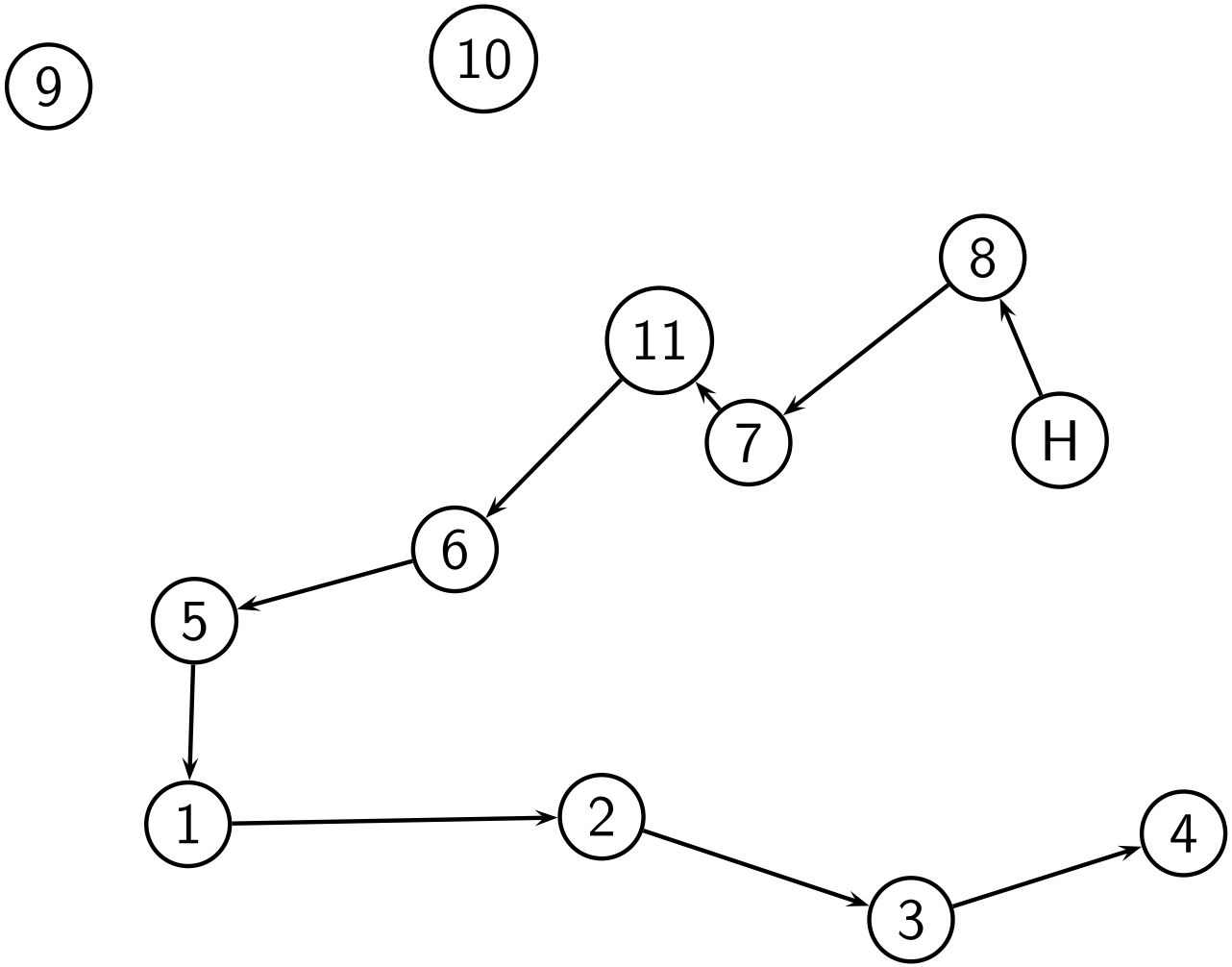
TSP: 12 cities



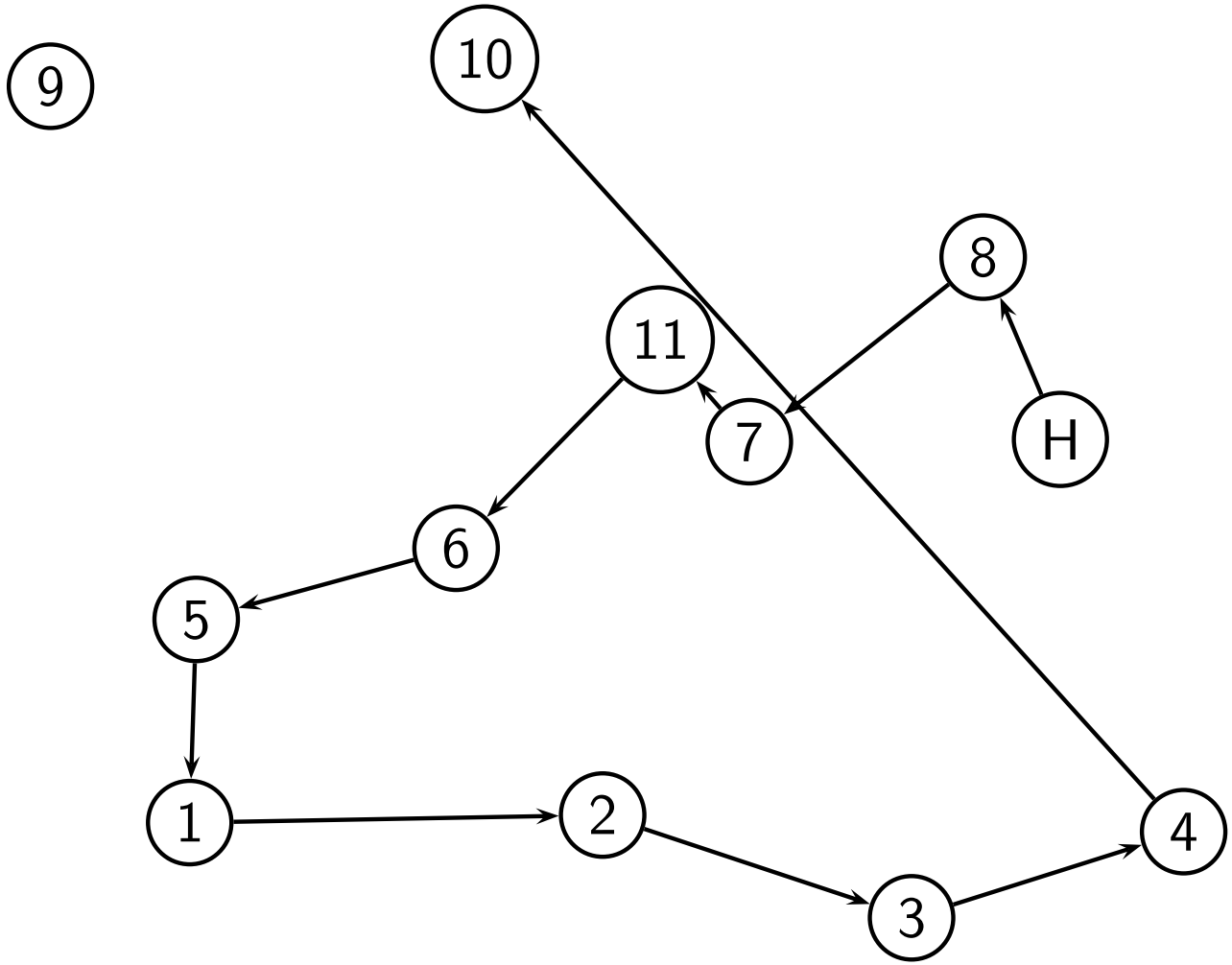
TSP: 12 cities



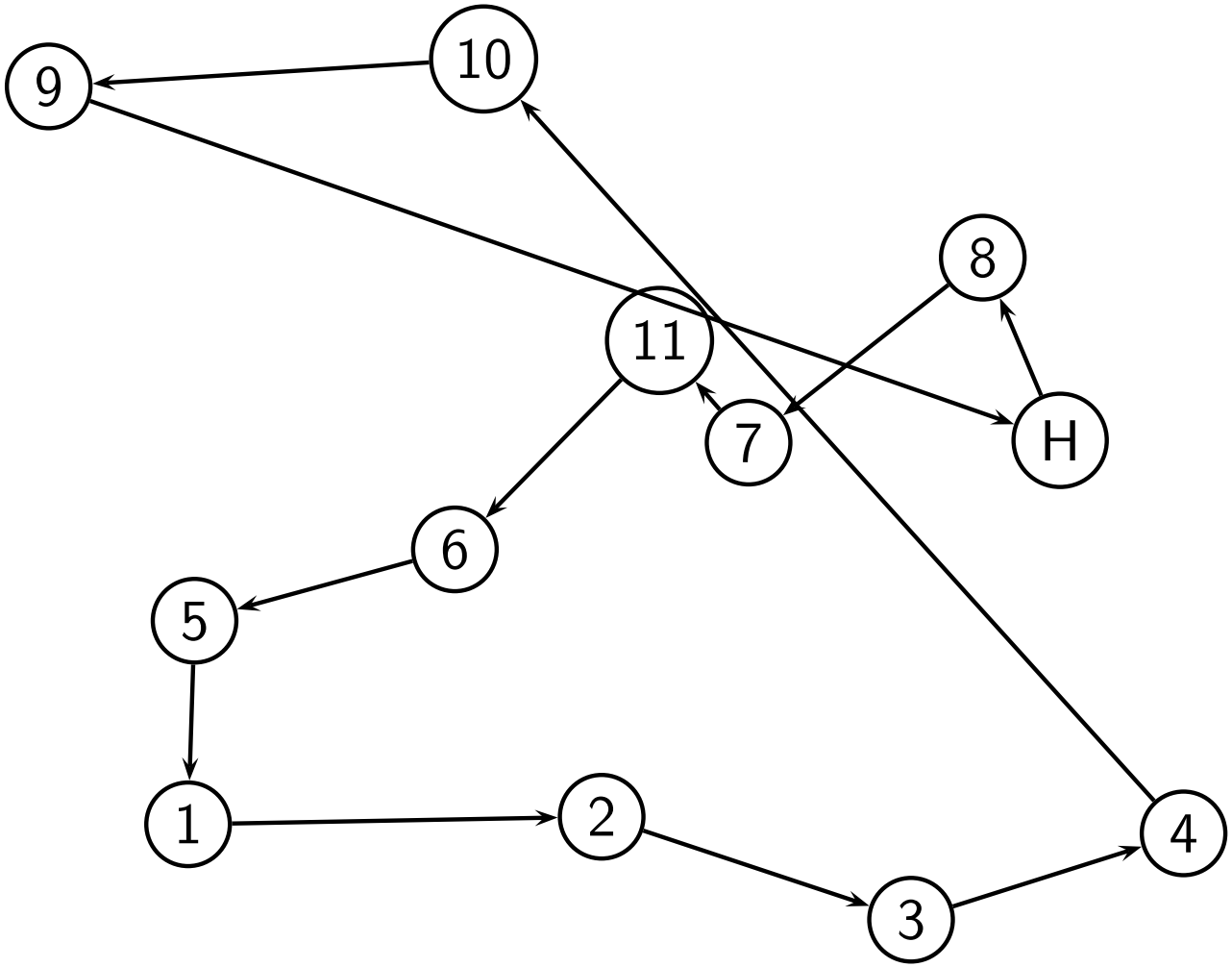
TSP: 12 cities



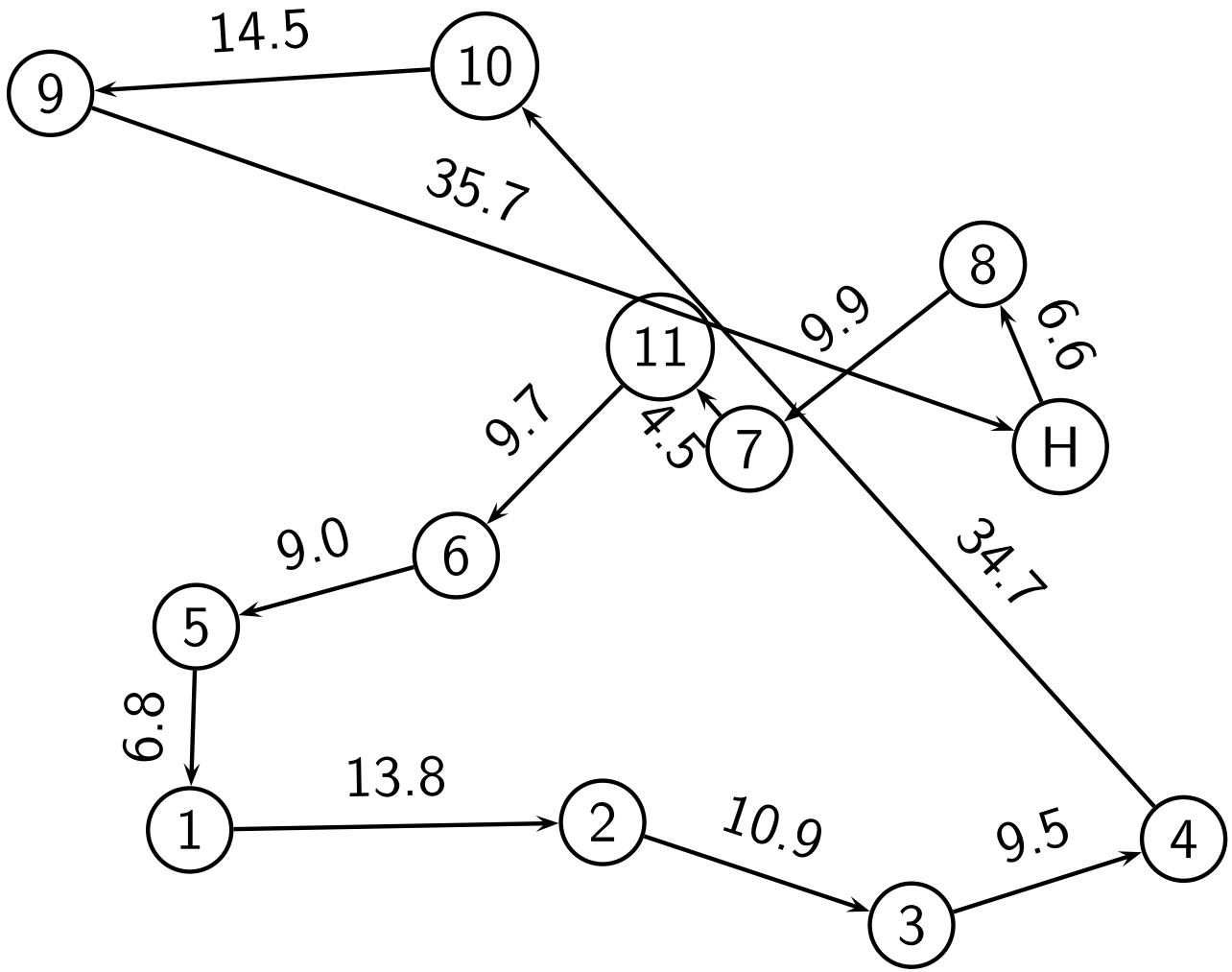
TSP: 12 cities



TSP: 12 cities



TSP: 12 cities



Length: 165.6

Integer optimization

Intuitive approach

- Solve the continuous relaxation.
- Round the solution.

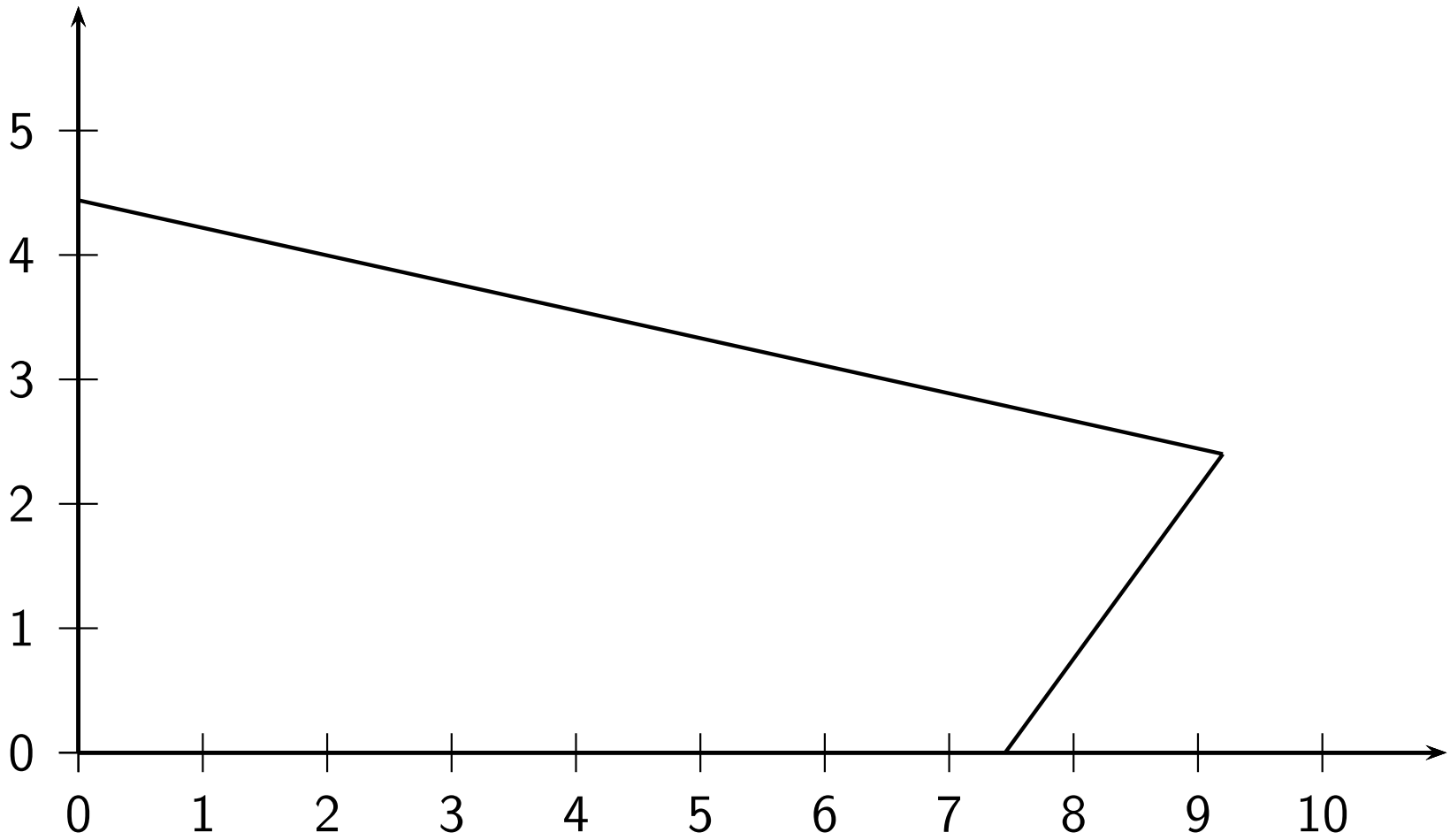
Example

$$\min_{x \in \mathbb{R}^2} -3x_1 - 13x_2$$

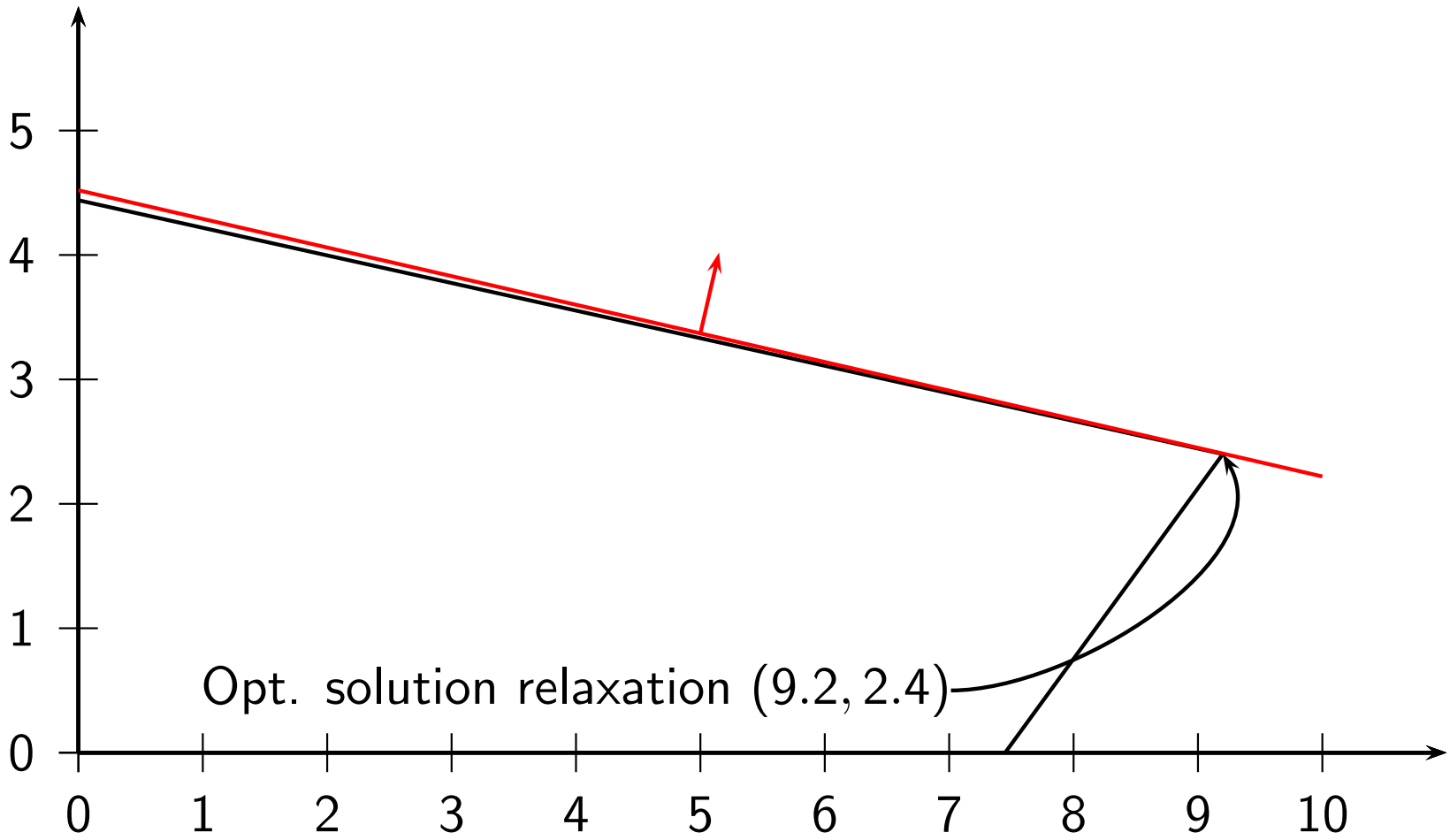
subject to

$$\begin{aligned} 2x_1 + 9x_2 &\leq 40 \\ 11x_1 - 8x_2 &\leq 82 \\ x_1, x_2 &\geq 0 \\ x_1, x_2 &\in \mathbb{N} \end{aligned}$$

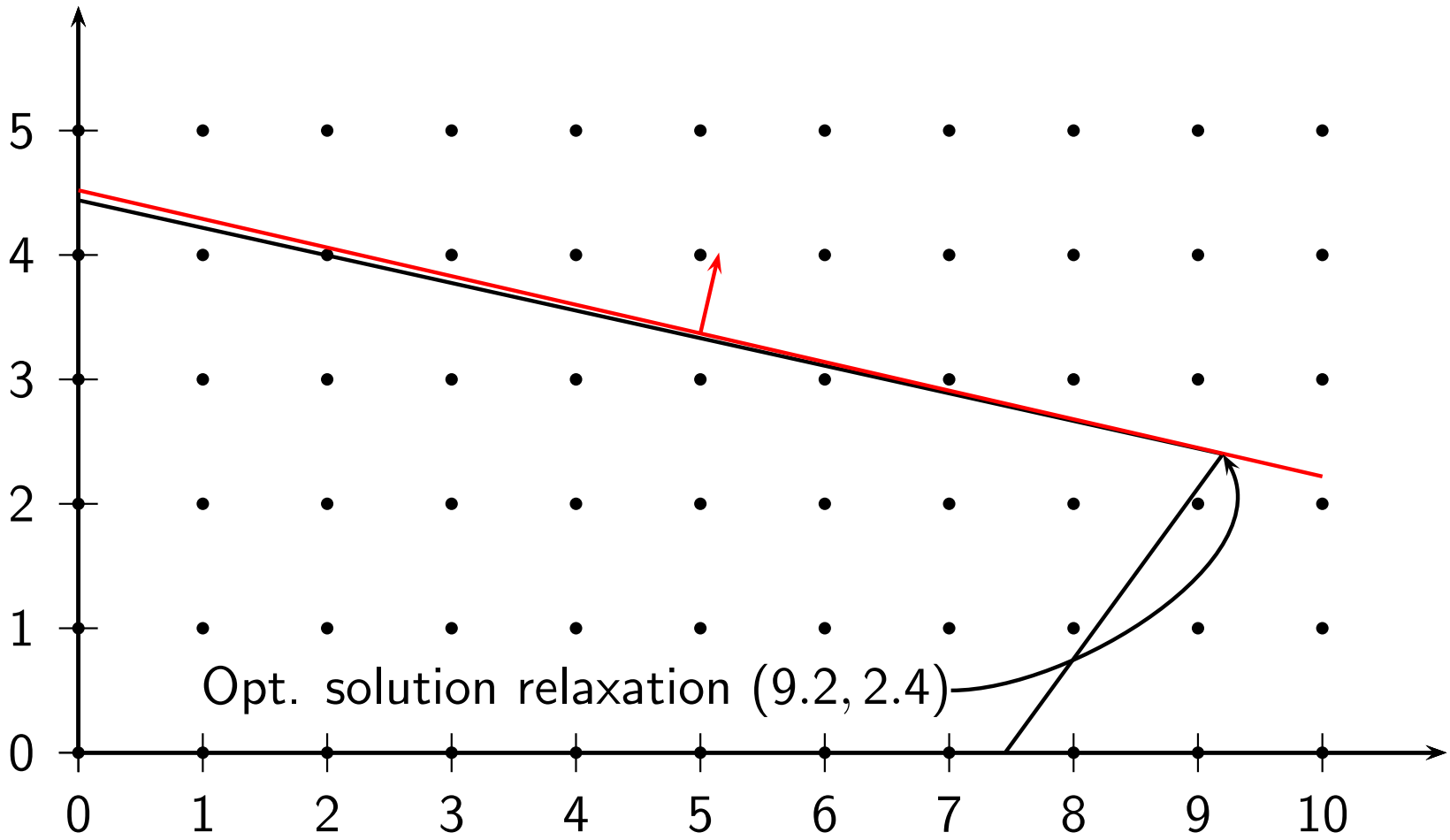
Relaxation: feasible set



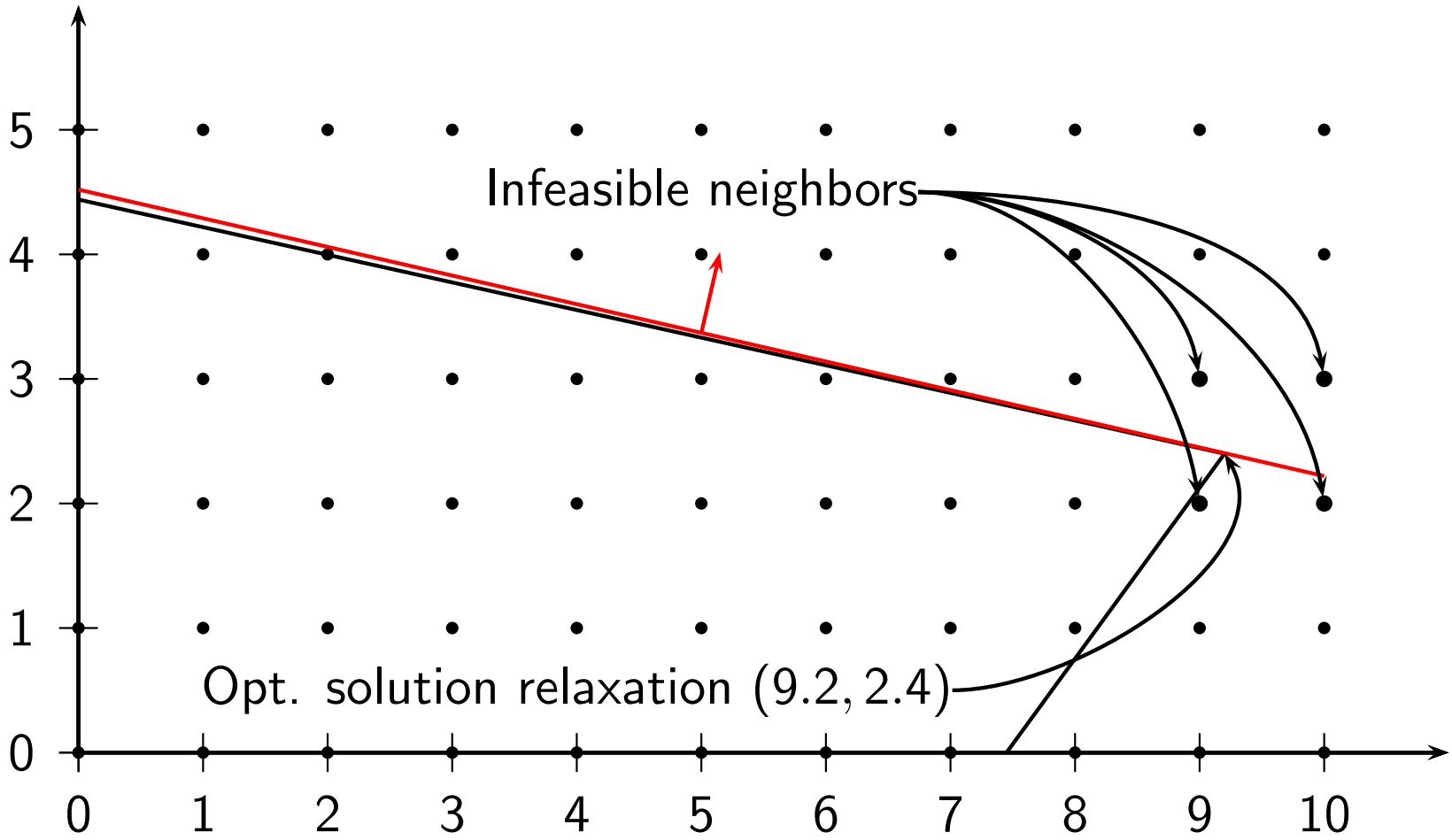
Optimal solution of the relaxation



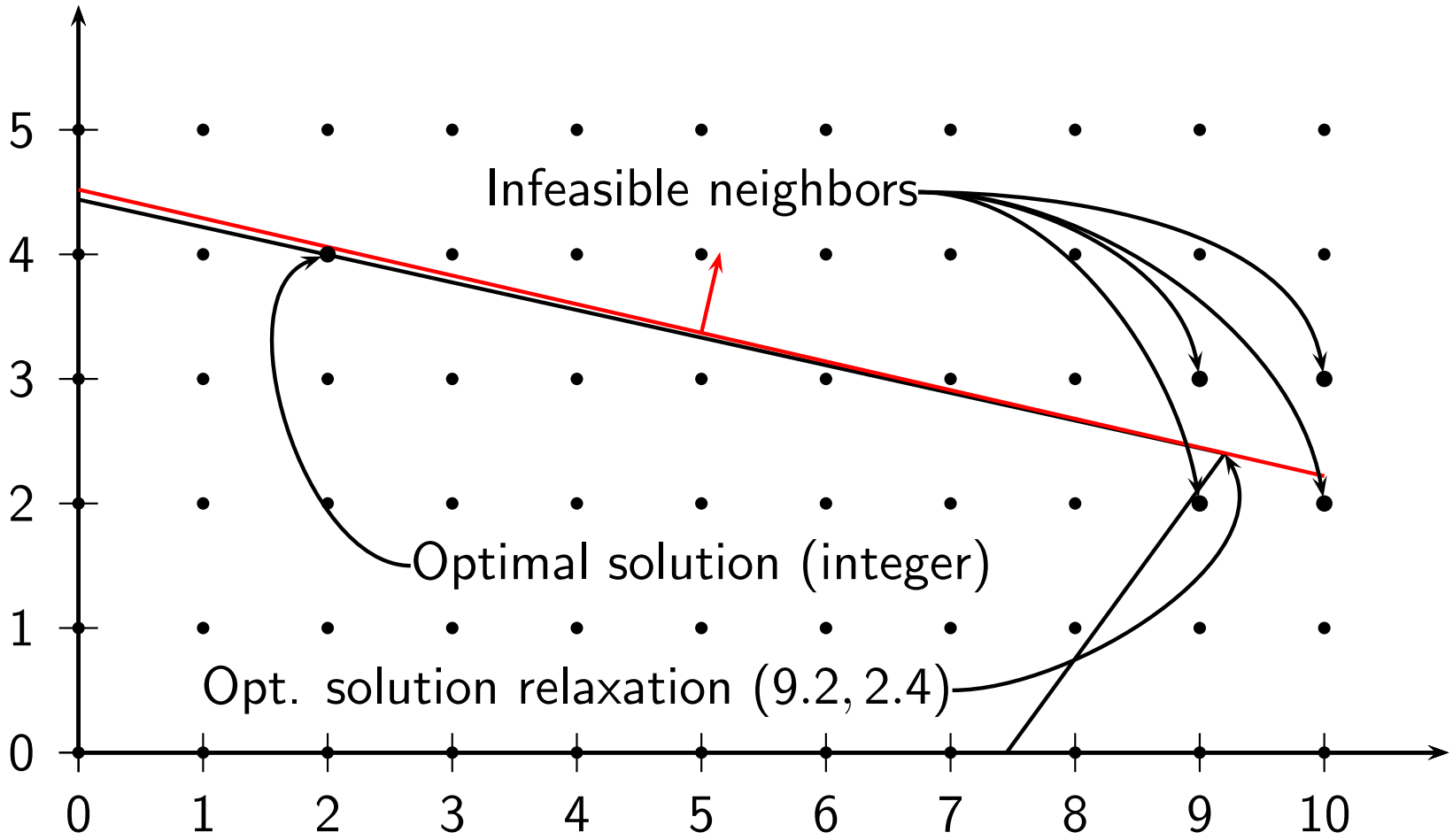
Integrality constraints



Infeasible neighbors



Solution of the integer optimization problem



Issues

- There are 2^n different ways to round. Which one to choose?
- Rounding may generate an infeasible solution.
- The rounded solution may be far from the optimal solution.

Optimization methods

Exact methods (branch and bound)

- Finds the optimal solution.
- Suffers from the curse of dimensionality.

Approximation algorithms

- Finds a sub-optimal solution.
- Guarantees a bound on the quality of the solution.
- Mainly used for theoretical purposes.

Heuristics

- Smart exploration of the solution space.
- No guarantee about optimality.
- Designed to mimic manual interventions.

Outline

Heuristics

Three main mechanisms

Exploration

- Objective: create operators to visit the solution space.
- Main concept: neighborhood.

Intensification

- Objective: improve an existing solution as much as possible.
- Main concepts: local search.

Diversification

- Objective: explore over regions of the solution space.
- Main concept: metaheuristics.

Neighborhood

Concept

- The feasible set is too large.
- At each iteration, restrict the optimization problem to a small feasible subset.
- Ideally, the small subset can be enumerated.
- Typically, it consists of solutions obtained from simple modifications of the current solution.
- The small subset is called a *neighborhood*.

Integer optimization: neighborhood

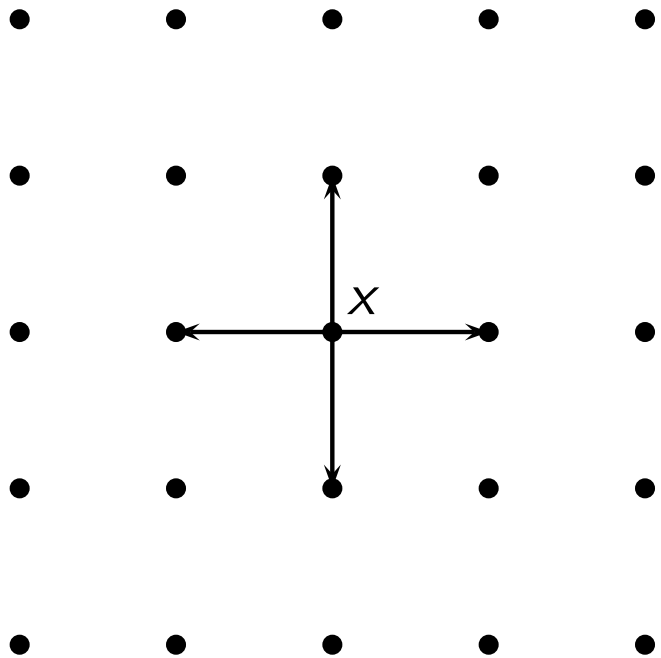
- Consider the current iterate $x \in \mathbb{Z}^n$.
- For each $k = 1, \ldots, n$, define 2 neighbors by increasing and decreasing the value of x_k by one unit.
- The neighbors y^{k+} and y^{k-} are defined as

$$y_i^{k+} = y_i^{k-} = x_i, \forall i \neq k, \quad y_k^{k+} = x_k + 1, \quad y_k^{k-} = x_k - 1.$$

- Example

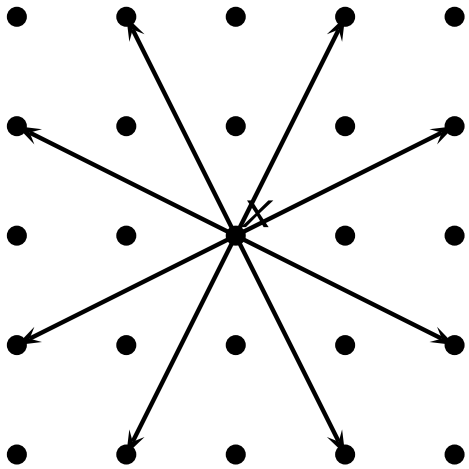
$$x = (3, 5, 2, 8) \quad y^{2+} = (3, 6, 2, 8) \quad y^{2-} = (3, 4, 2, 8)$$

Integer optimization: neighborhood



Integer optimization: neighborhood

- The concept of neighborhood is fairly general.
- It must be defined based on the structure of the problem.
- Creativity is required here.



Local search

- Consider the integer optimization problem

$$\min_{x \in \mathbb{Z}^n} f(x)$$

subject to

$$x \in \mathcal{F}.$$

- Consider the neighborhood structure $V(x)$, where $V(x)$ is the set of neighbors of x.
- At each iteration k, consider the neighbors in $V(x_k)$ one at a time.
- For each $y \in V(x_k)$, if $f(y) < f(x_k)$, then $x_{k+1} = y$ and proceed to the next iteration.
- If $f(y) \geq f(x_k)$, $\forall y \in V(x_k)$, x_k is a local minimum. Stop.

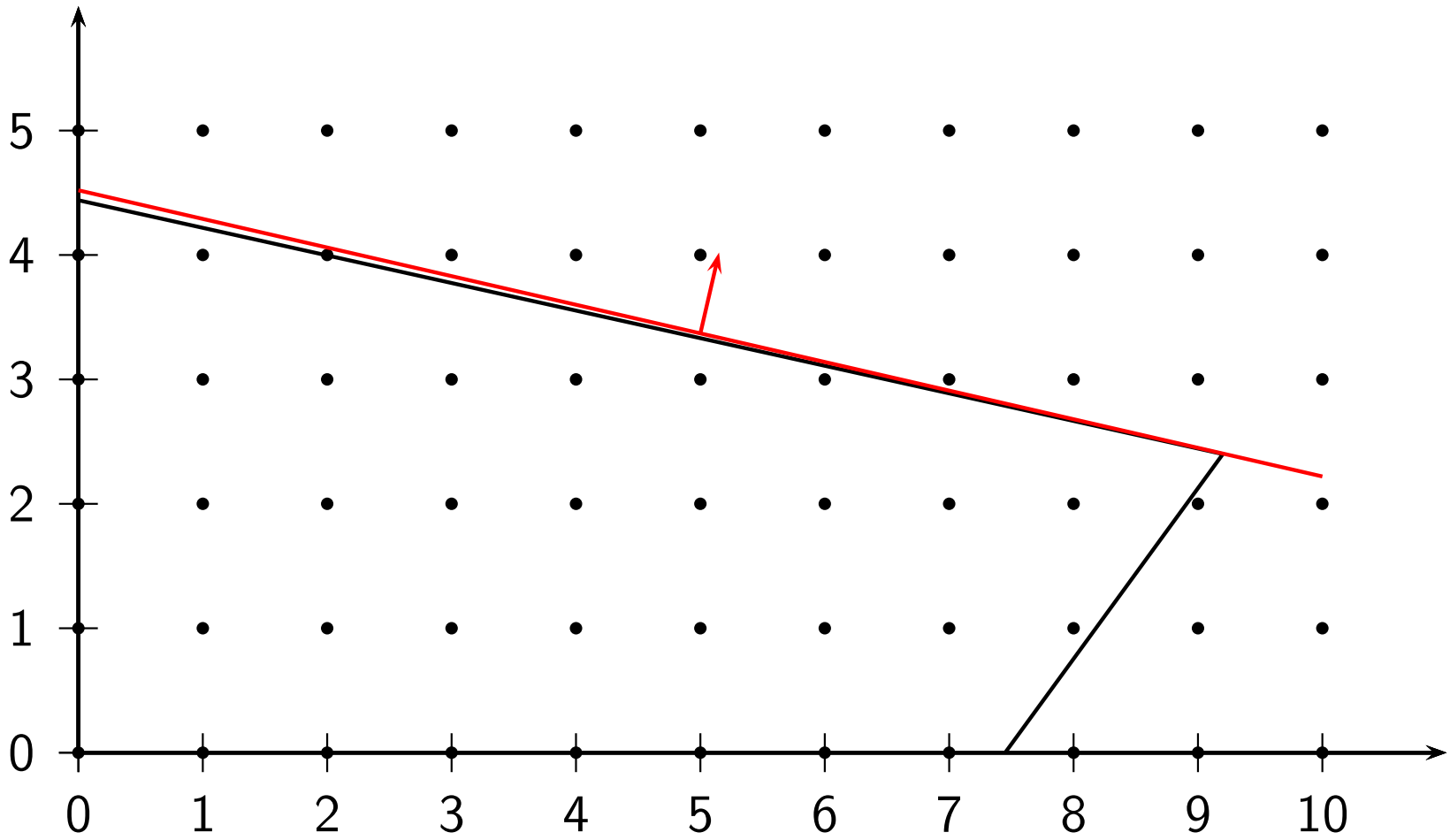
Local search: example

$$\min_{x \in \mathbb{R}^2} -3x_1 - 13x_2$$

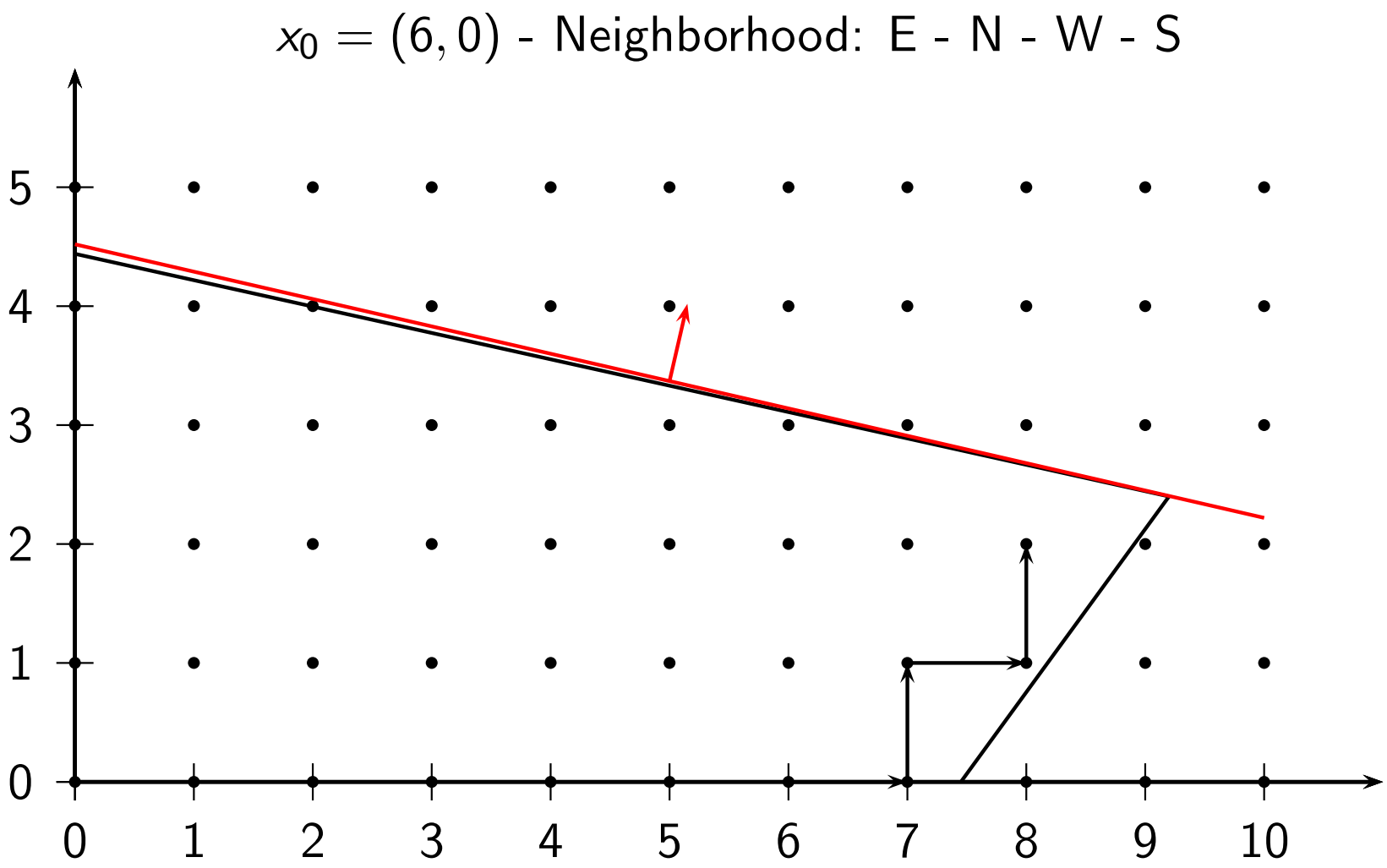
subject to

$$\begin{aligned} 2x_1 + 9x_2 &\leq 40 \\ 11x_1 - 8x_2 &\leq 82 \\ x_1, x_2 &\in \mathbb{N} \end{aligned}$$

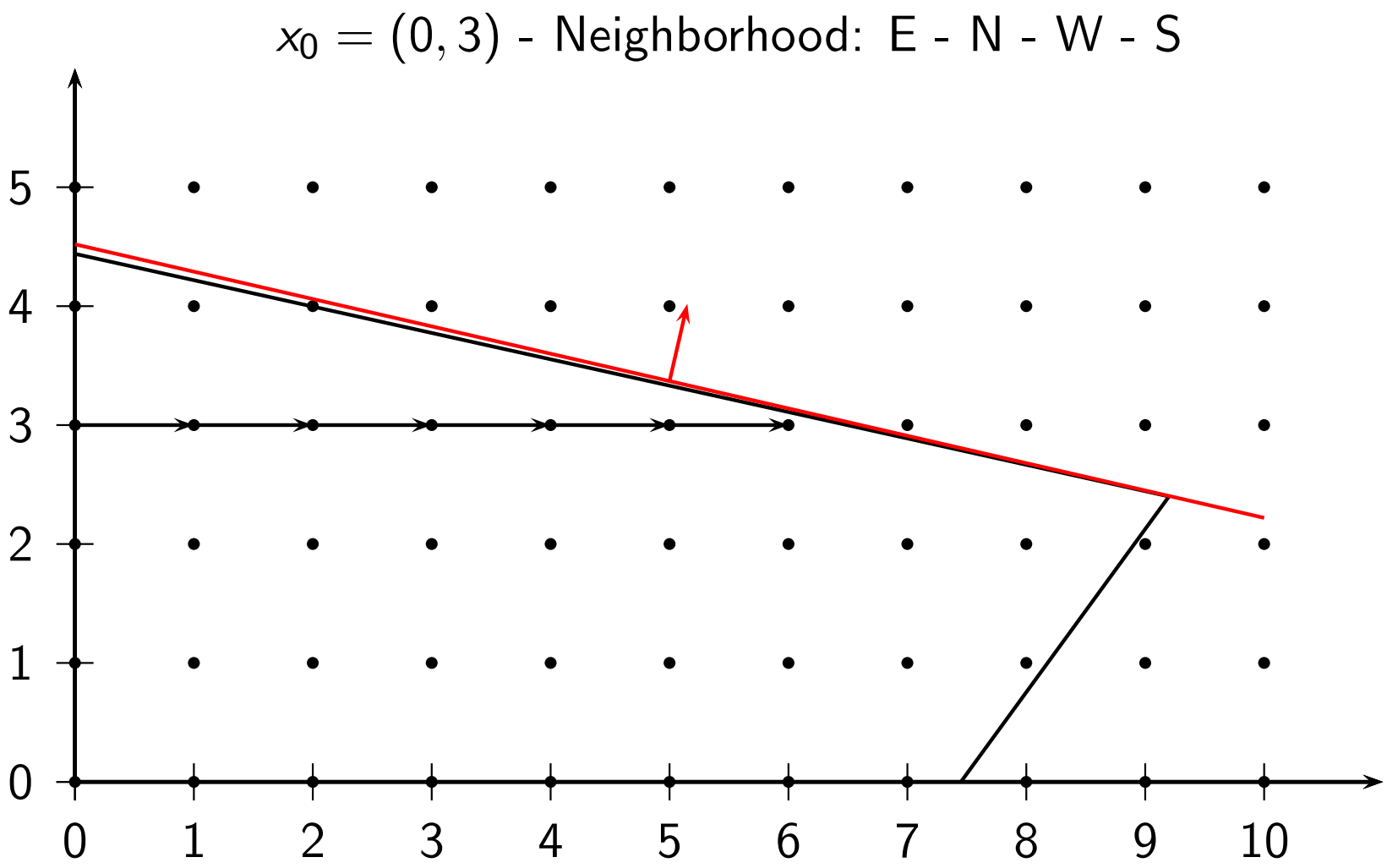
Local search: example



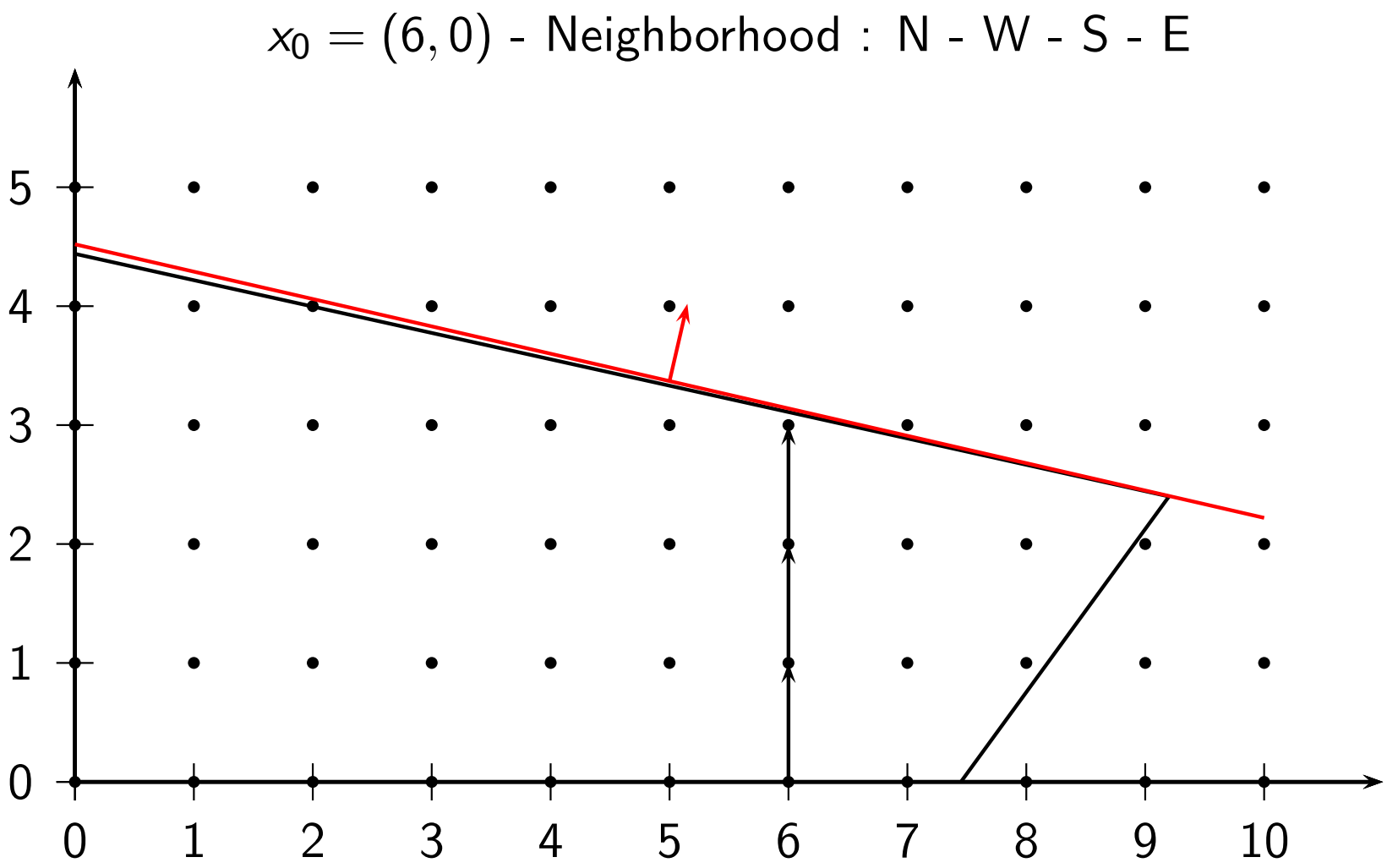
Local search: example



Local search: example



Local search: example



Local search: comments

- The algorithm stops at a local minimum, that is a solution better than all its neighbors.
- The outcome depends on the starting point and the structure of the neighborhood.
- The neighborhood must be sufficiently large to increase the chances of improvement, and sufficiently small to avoid a lengthy enumeration.
- Example of a neighborhood too small: one neighbor at the west.
- Example of a neighborhood too large: each feasible point is in the neighborhood.
- It is good practice to use symmetric neighborhoods:

$$y \in V(x) \Longleftrightarrow x \in V(y).$$

The knapsack problem

$$\max_{x \in \{0,1\}^n} u^T x$$

subject to

$$w^T x \leq W.$$

The knapsack problem: neighborhood

- Current solution: for each item i, $x_i = 0$ or $x_i = 1$.
- Neighbor solution: select an item j, and change the decision: $x_j \leftarrow 1 - x_j$.
- Warning: check feasibility.
- Generalization: neighborhood of size k: select k items, and change the decision for them (checking feasibility).

The knapsack problem: neighborhood

- A neighborhood of size k modifies k variables.
- Number of neighbors:

$$\frac{n!}{k!(n-k)!}$$

- $k = 1$: n neighbors.
- $k = n$: 1 neighbor.

TSP: 2-OPT neighborhood

2-OPT

- Select two cities.
- Swap their position in the tour.
- Visit all intermediate cities in reverse order.

Example

Current tour:

A–B–C–D–E–F–G–H–A

Exchange C and G to obtain

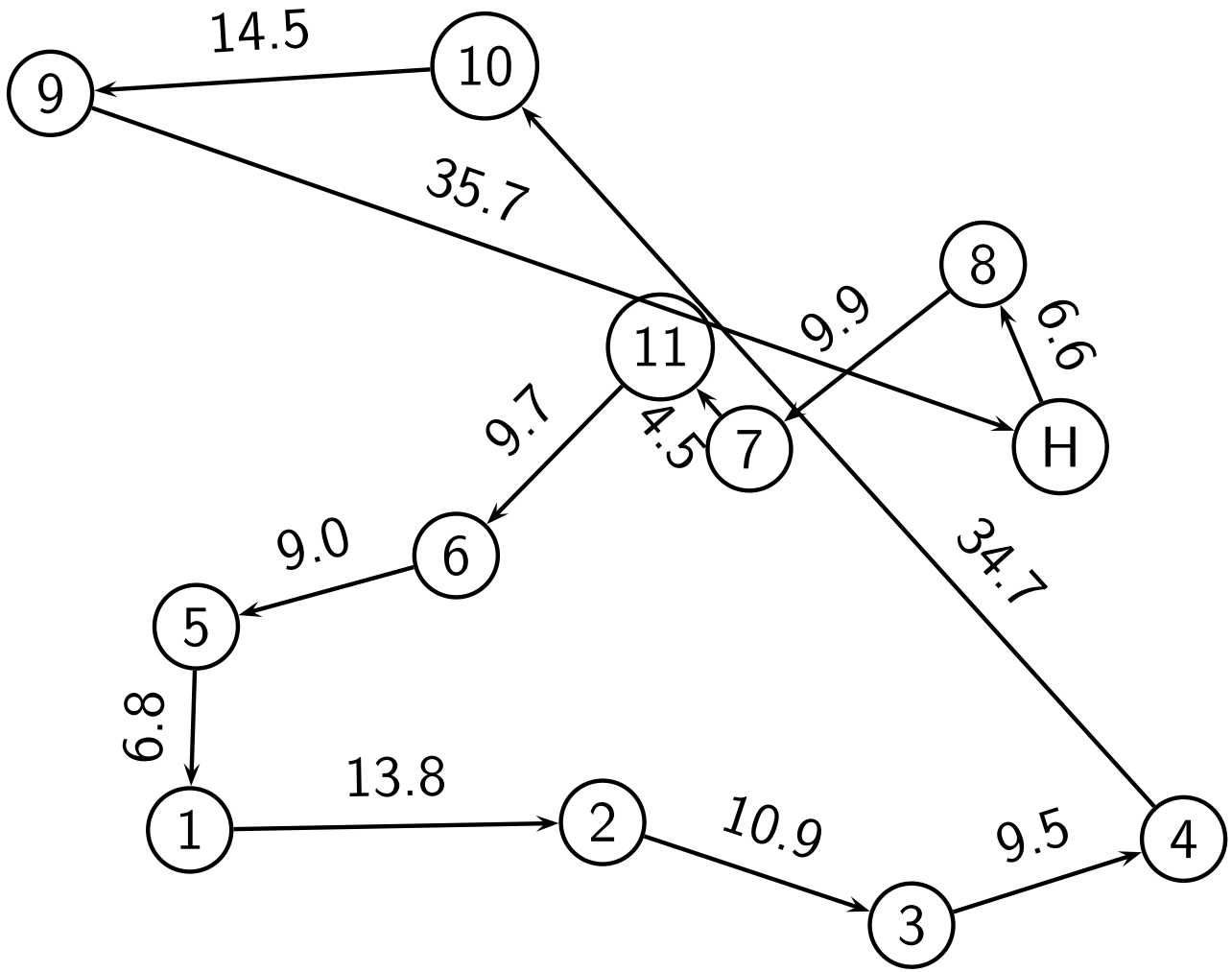
A–B–G–F–E–D–C–H–A.

Neighborhood: 2-OPT(1,9)

Example

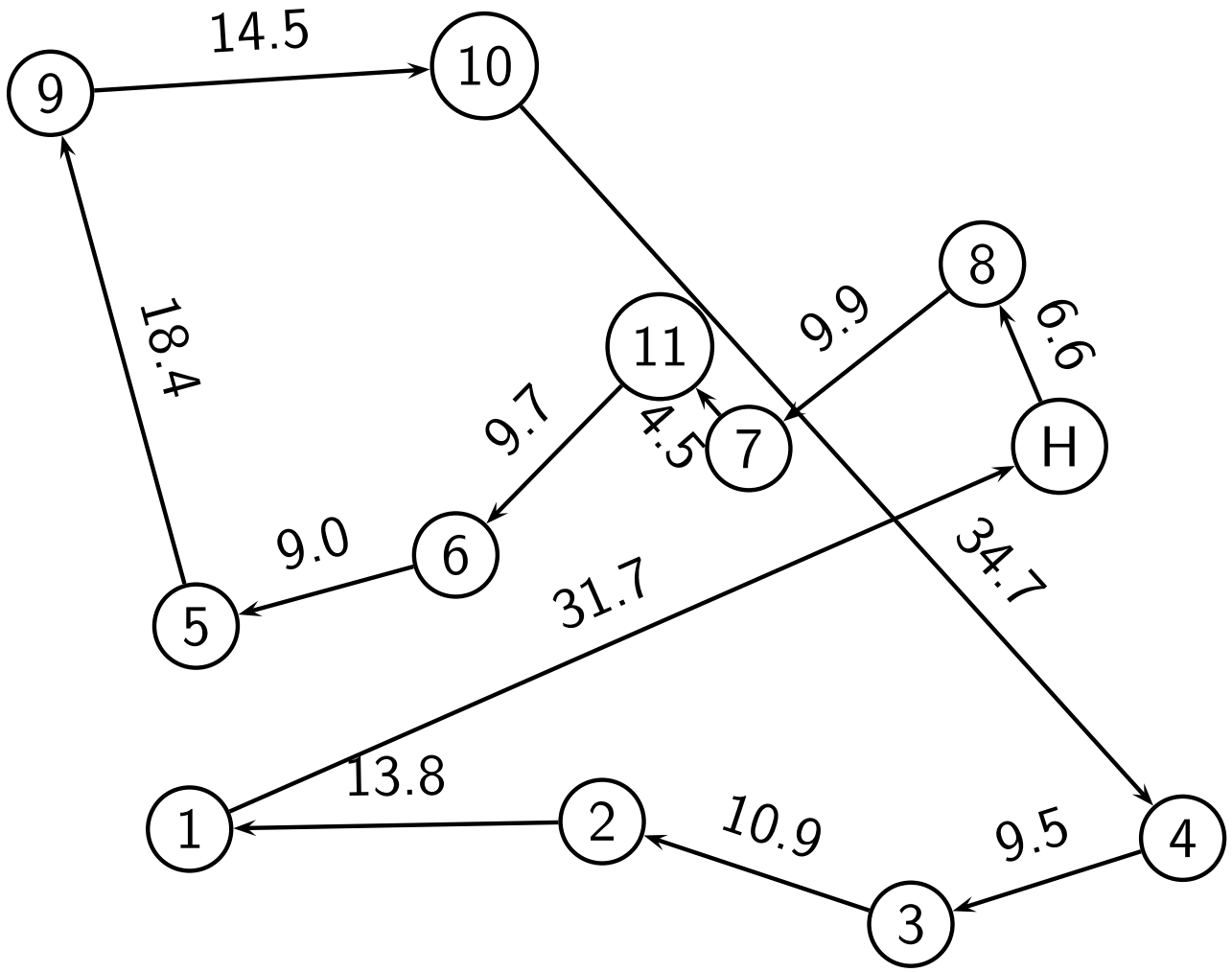
- Try to improve the solution using 2-OPT swapping 1 and 9.
- Before: H–8–7–11–6–5–1–2–3–4–10–9–H (length: 165.6)
- After : H–8–7–11–6–5–9–10–4–3–2–1–H (length: 173.3)
- No improvement.

Neighborhood: 2-OPT(1,9)



Length: 165.6

Neighborhood: 2-OPT(1,9)

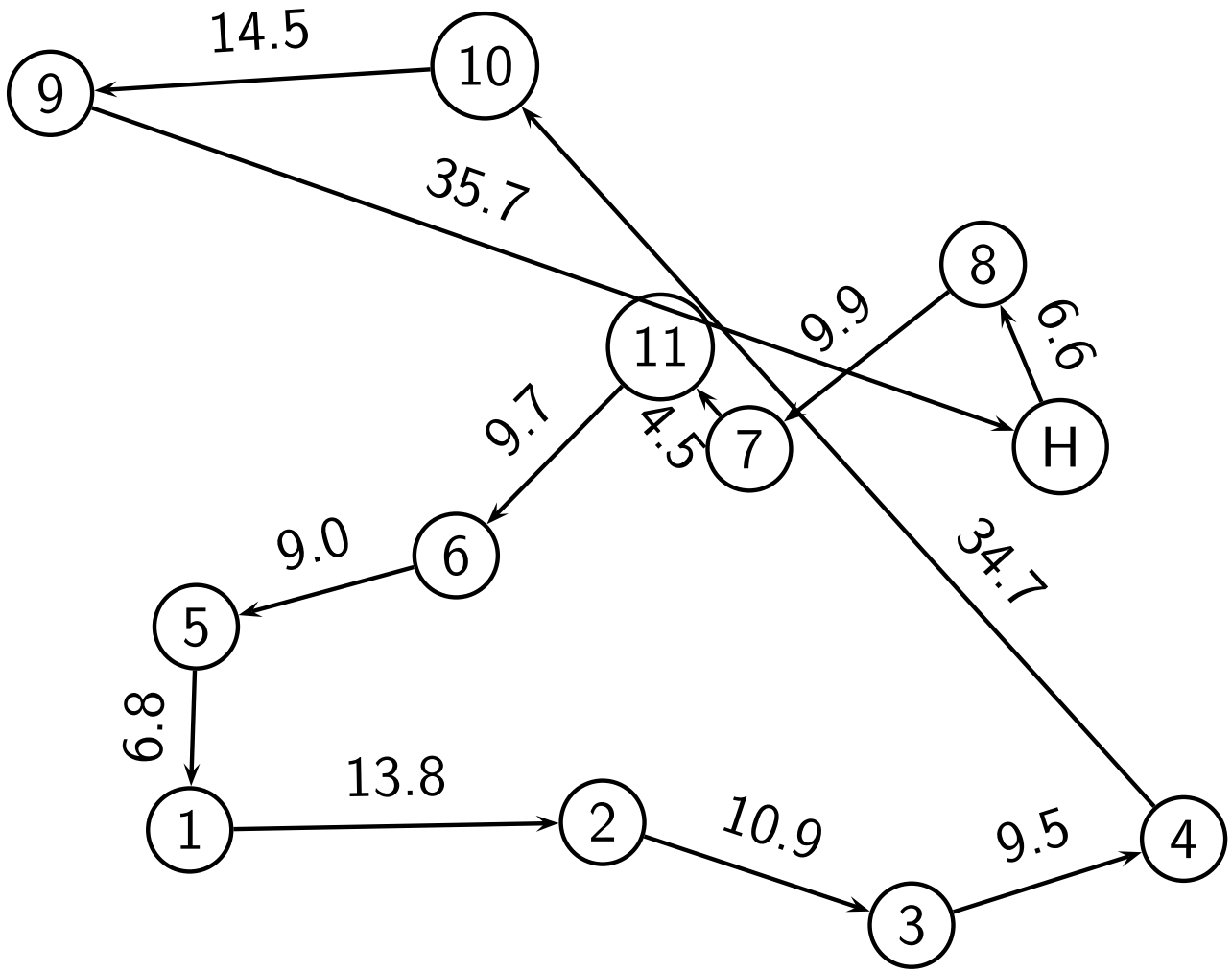


Length: 173.3

Local search

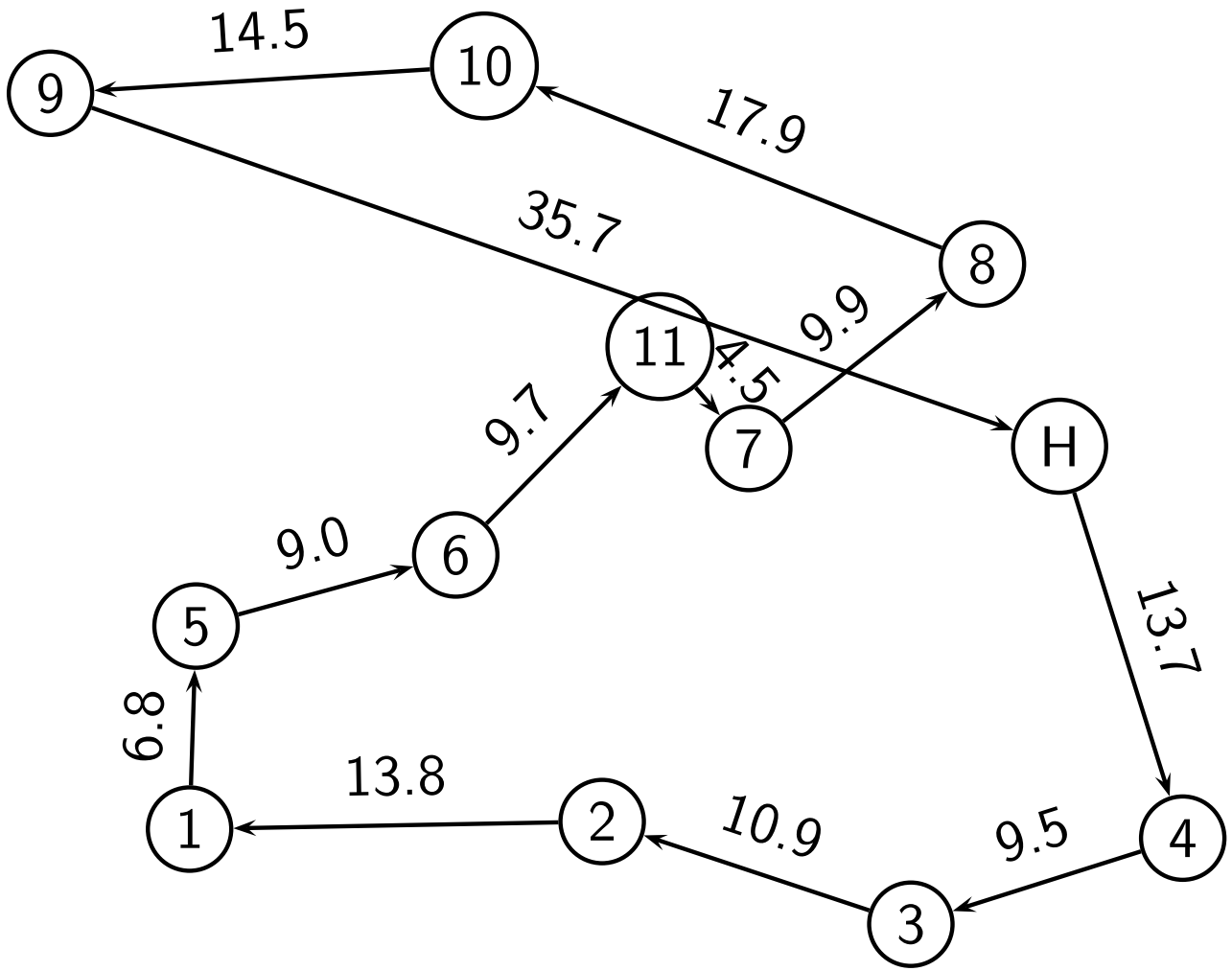
- Consider each pair of nodes.
- Apply 2-OPT.
- Select the best tour.

Current tour



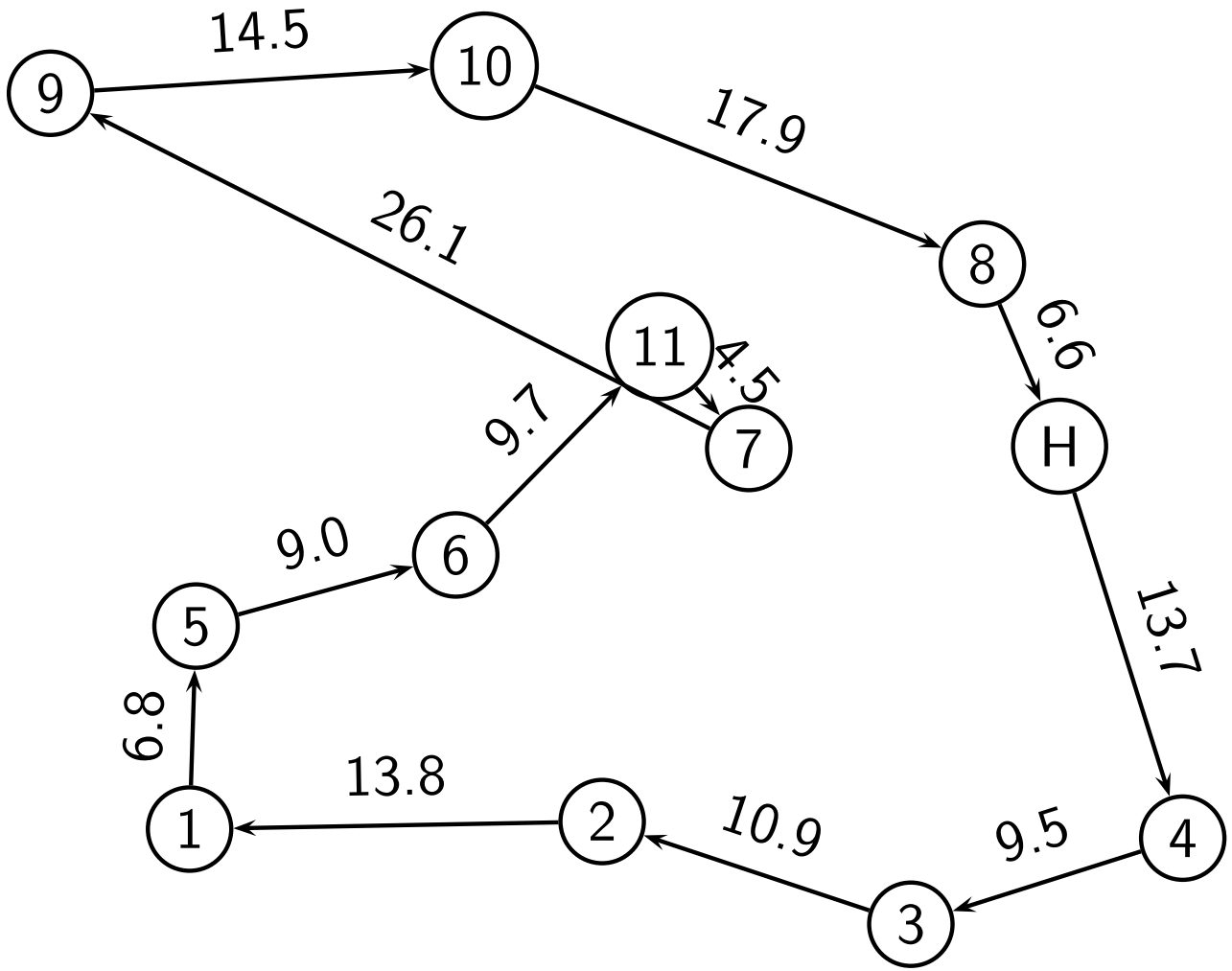
Length: 165.6

Best neighbor: 2-OPT(8,4)



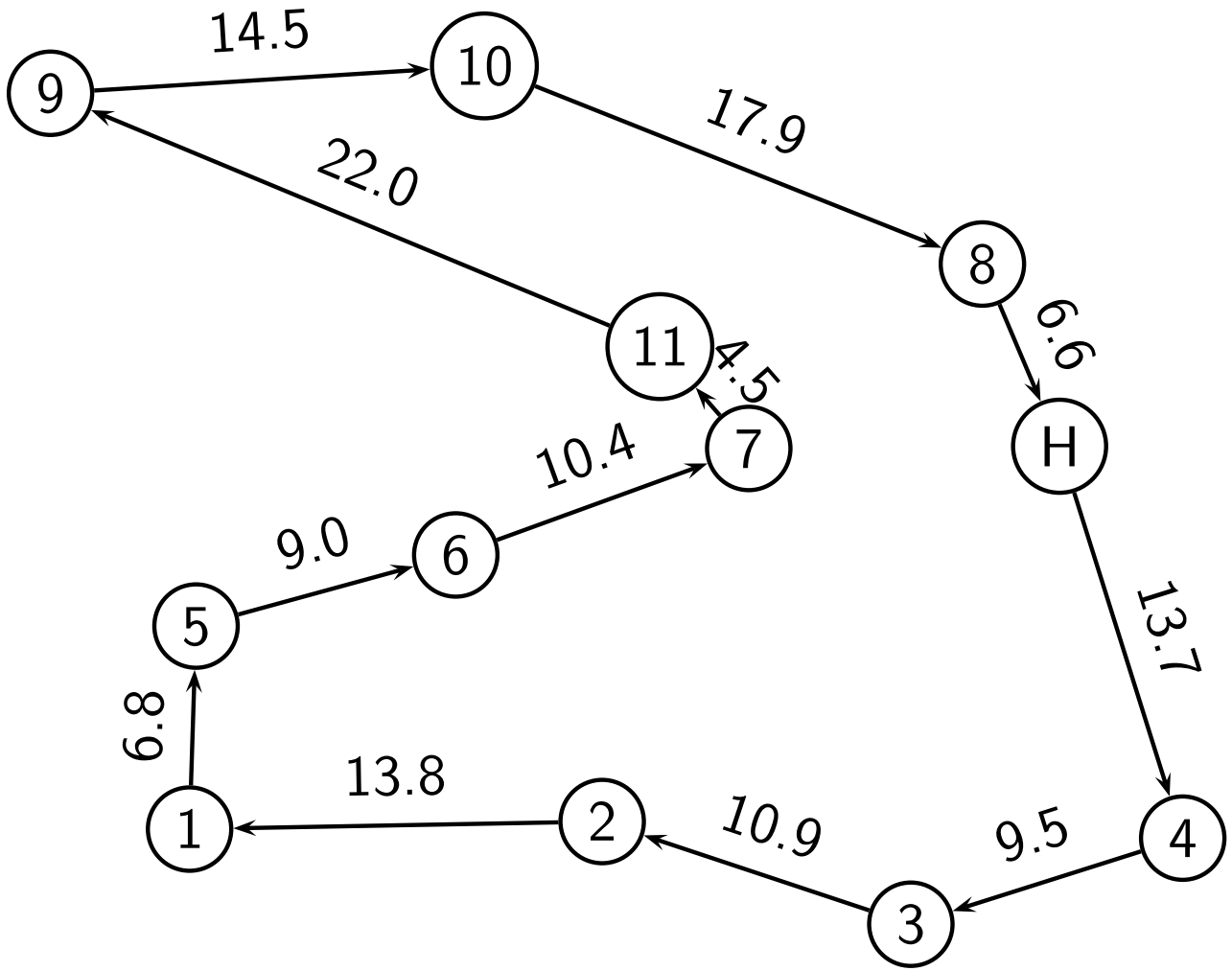
Length: 155.8

Best neighbor: 2-OPT(8,9)



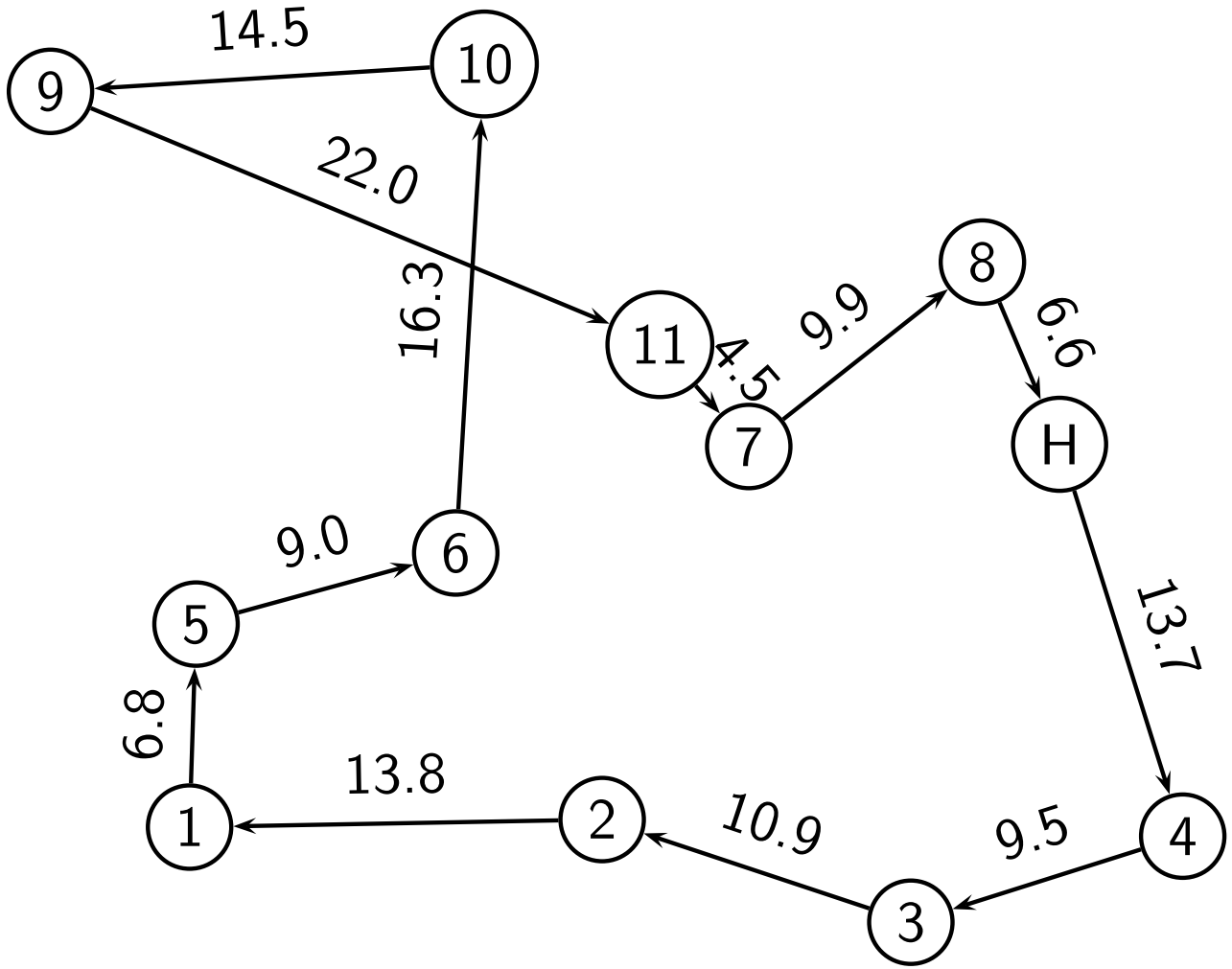
Length: 143.0

Best neighbor: 2-OPT(11,7)



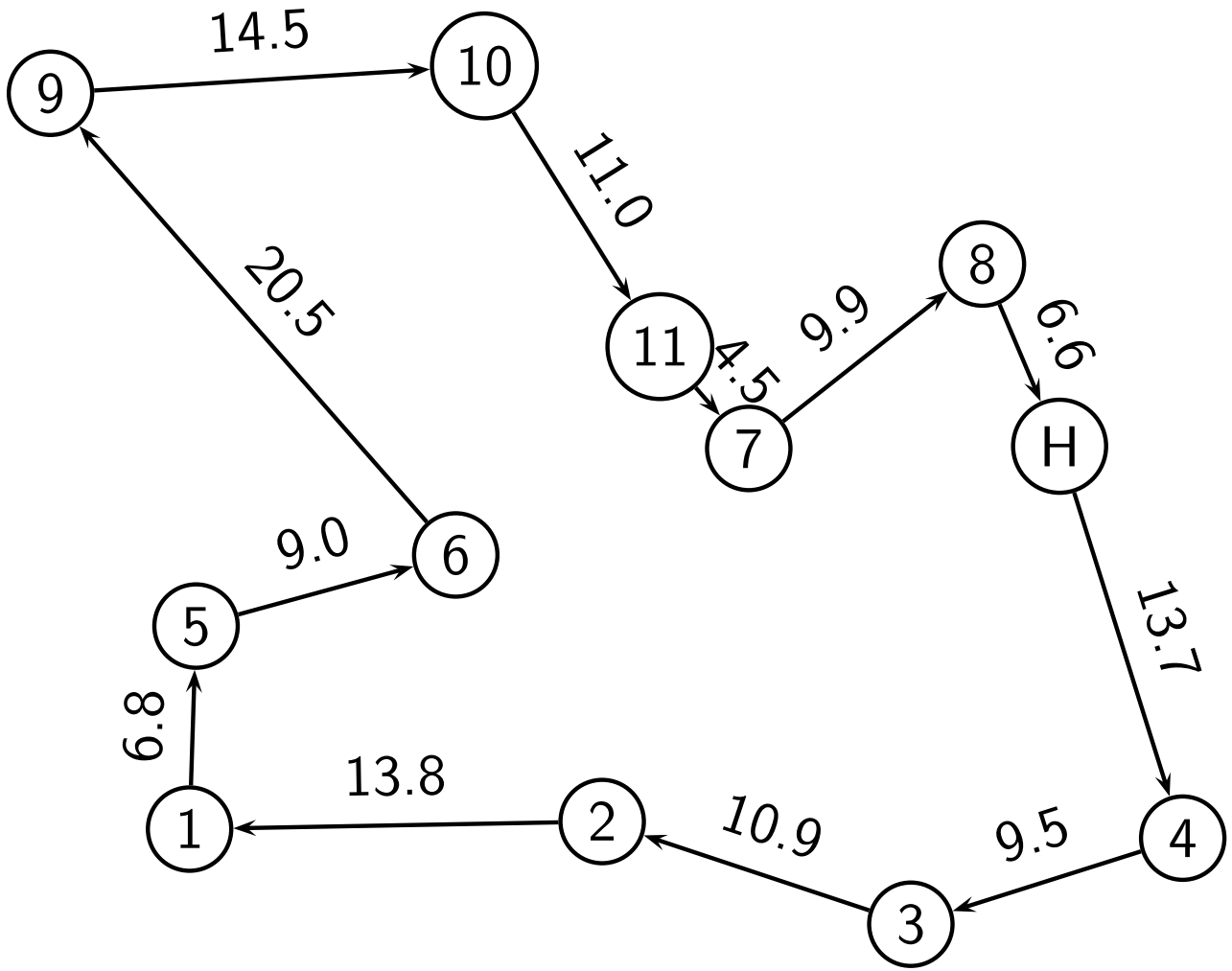
Length: 139.5

Best neighbor: 2-OPT(7,10)



Length: 137.5

Best neighbor: 2-OPT(10,9)



Length: 130.7

Variable Neighborhood Search

- aka VNS
- Idea: consider several neighborhood structures.
- When a local optimum has been found for a given neighborhood structure, continue with another structure.

VNS: method

Input
- V_1, V_2, ..., V_K neighborhood structures.
- Initial solution x_0.

Initialization
- $x_c \leftarrow x_0$
- $k \leftarrow 1$

Iterations Repeat
- Apply local search from x_c using neighborhood V_k

$$x^+ \leftarrow LS(x_c, V_k)$$

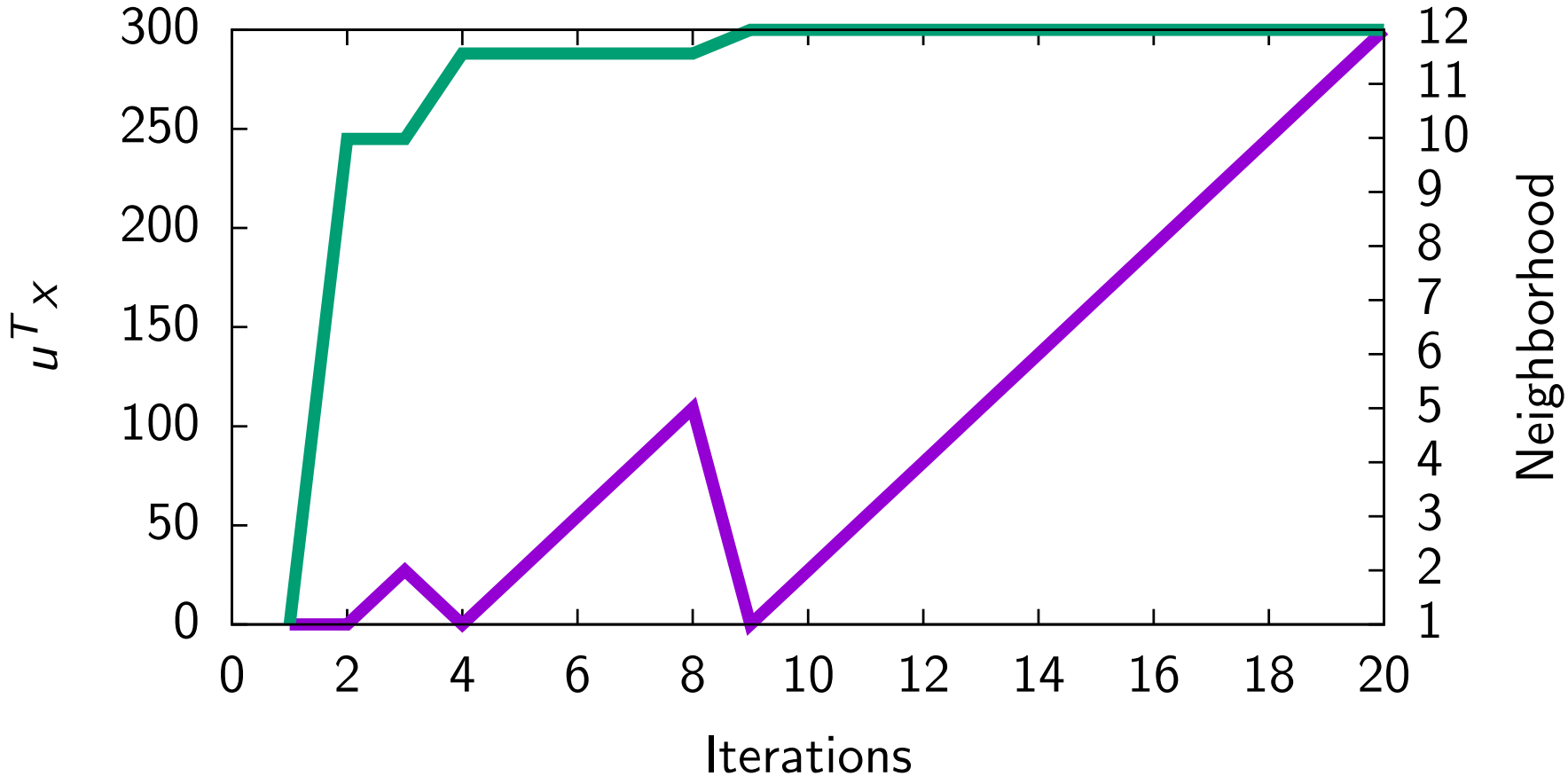
- If $f(x^+) < f(x_c)$, then $x_c \leftarrow x^+$, $k \leftarrow 1$.
- Otherwise, $k \leftarrow k + 1$.

Until $k = K$.

VNS: example for the knapsack problem

- Neighborhood of size k: modify k variables.
- Local search: current iterate: x_c
  - randomly select a neighbor x^+
  - if $w^T x^+ \leq W$ and $u^T x^+ > u^T x_c$, then $x_c \leftarrow x^+$
- Repeat 1000 times, for any k.
- The complexity is therefore independent of k.

VNS: example for the knapsack problem



Simulated annealing

Analogy with metallurgy

- Heating a metal and then cooling it down slowly improves its properties.
- The atoms take a more solid configuration.

In optimization:

- Local search can both decrease and increase the objective function.
- At "high temperature", it is common to increase.
- At "low temperature", increasing happens rarely.
- Simulated annealing: slow cooling = slow reduction of the probability to increase.

Simulated annealing

Modify the local search.
For the sake of simplicity: consider a neighborhood structure containing only feasible solutions.
Let x_k be the current iterate

- Select $y \in V(x_k)$.
- If $f(y) \leq f(x_k)$, then $x_{k+1} = y$.
- Otherwise, $x_{k+1} = y$ with probability

$$e^{-\frac{f(y)-f(x_k)}{T}}$$

with $T > 0$.

Concretely, draw r between 0 and 1.
Accept y as next iterate if

$$e^{-\frac{f(y)-f(x_k)}{T}} > r$$

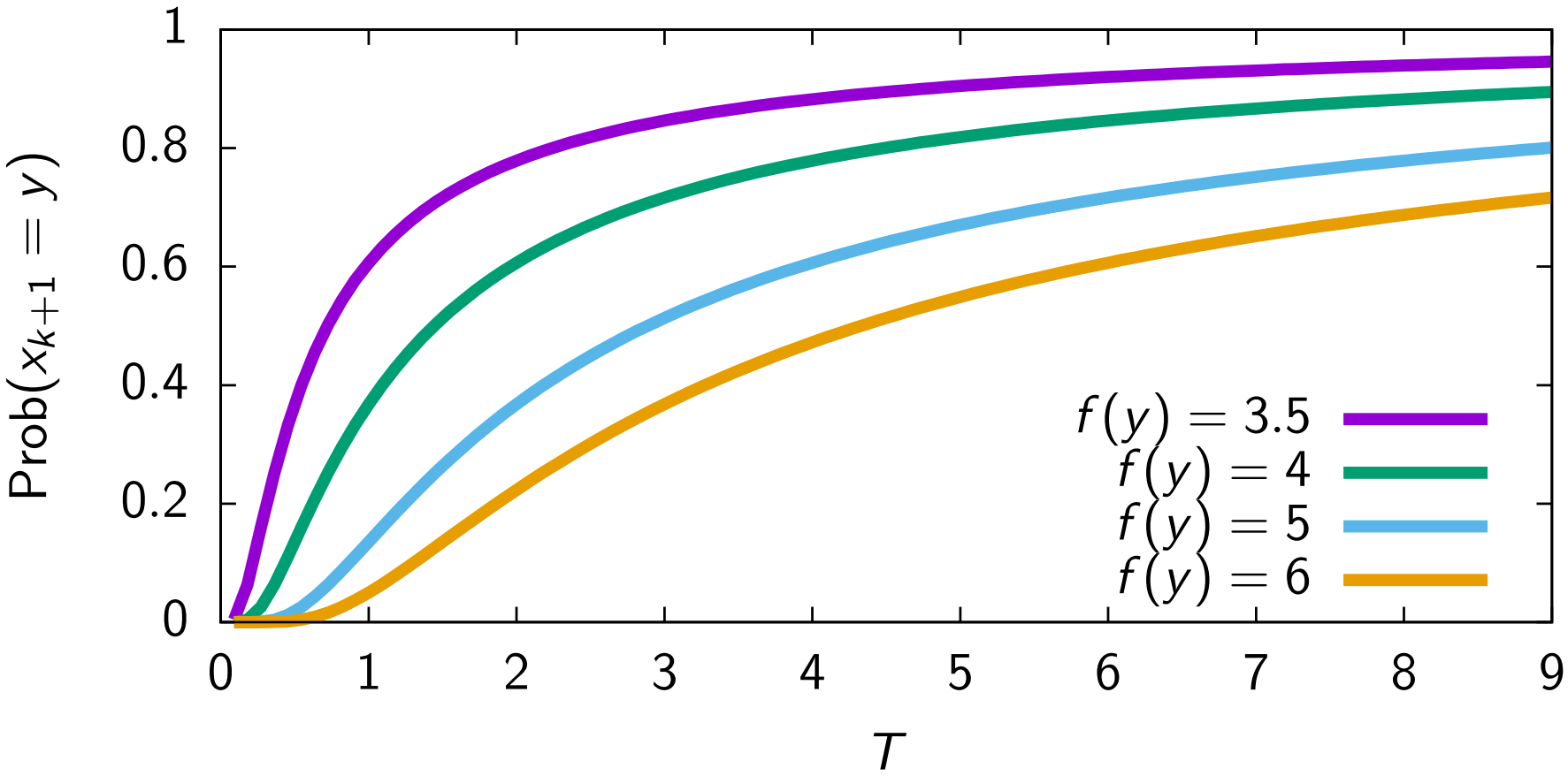
Simulated annealing

$$\text{Prob}(x_{k+1} = y) = \begin{cases} 1 & \text{if } f(y) \leq f(x_k) \\ e^{-\frac{f(y)-f(x_k)}{T}} & \text{if } f(y) > f(x_k) \end{cases}$$

- If T is high (hot temperature), high probability to increase.
- If T is low, almost only decreases.

Simulated annealing

Example : $f(x_k) = 3$



Simulated annealing

- In practice, start with high T for flexibility.
- Then, decrease T progressively.

Simulated annealing

Input
- Initial solution x_0
- Initial temperature T_0, minimum temperature T_f
- Neighborhood structure $V(x)$
- Maximum number of iterations K

Initialize $x_c \leftarrow x_0$, $x^* \leftarrow x_0$, $T \leftarrow T_0$

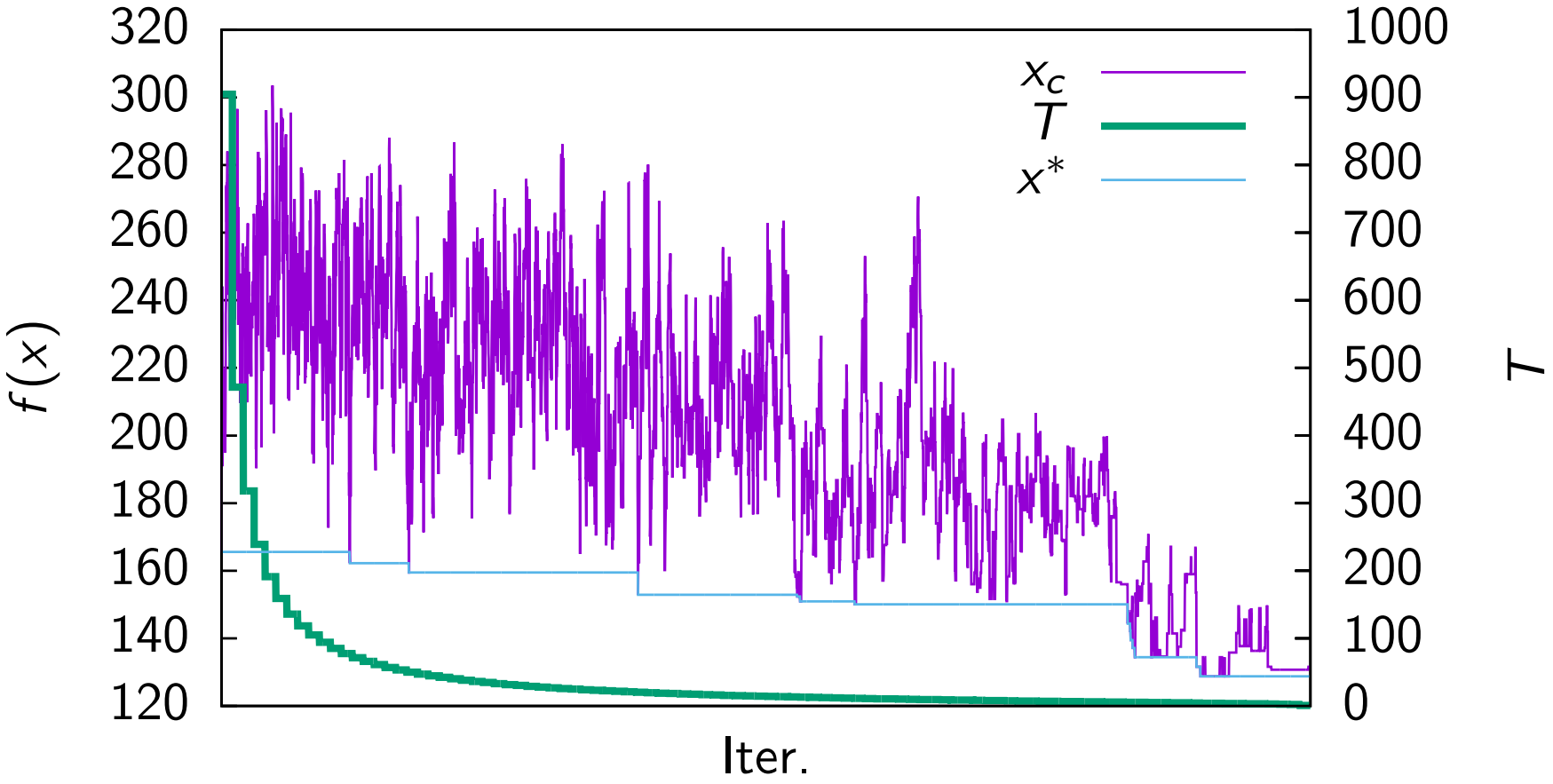
Simulated annealing

Repeat $k \leftarrow 1$

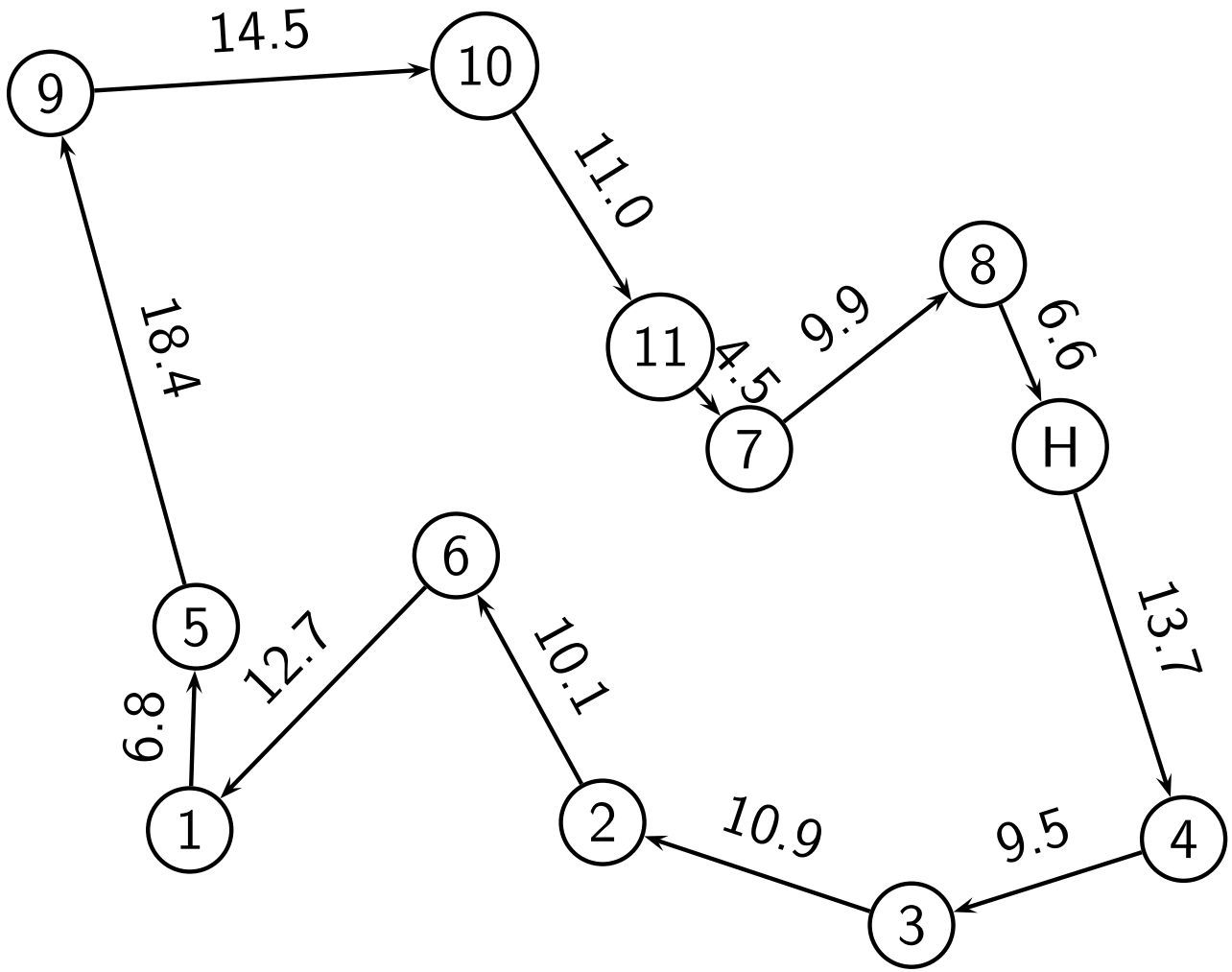
- While $k < K$
  - Randomly select a neighbor $y \in V(x_c)$
  - $\delta \leftarrow f(y) - f(x_c)$
  - If $\delta < 0$, $x_c = y$.
  - Otherwise, draw r between 0 and 1
    - If $r < exp(-\delta/T)$, then $x_c = y$
  - If $f(x_c) < f(x^*)$, $x^* = x_c$.
  - $k \leftarrow k + 1$
- Reduce T

Until $T \leq T_f$

Example: traveling salesman problem



Best solution found



Length : 128.8

Practical comments

- Parameters must be tuned.
- In particular, the reduction rate of the temperature must be specified.
  - Let δ_t be a typical increase of the objective function.
  - In the beginning, we want such an increase to be accepted with probability p_0 (e.g. $p_0 = 0.999$)
  - At the end, we want such an increase to be accepted with probability p_f (e.g. $p_f = 0.00001$)
  - We allow for M updates of the temperature. So, for $m = 0, \ldots, M$,

$$T = -\frac{\delta_t}{\ln(p_0 + \frac{p_f - p_0}{M} m)}$$

Heuristics: general framework

Exploration

Neighborhood

Intensification

Local search

Diversification

Escape from local minima

Exploration: comments

- Neighborhood structure
- It is a "vehicle" to explore the solution space.
- Must be able to (potentially) reach any solution
- Must be tailored to the problem

Intensification: comments

- Local search.
- Exploit the neighborhood structure to find better solutions.
- Many variants are possible:
  - exhaustive search: evaluate all neighbors
  - limited search: evaluate a given number of neighbors
  - randomized search: select the neighbors randomly

Diversification: comments

How to avoid being blocked in local minimum?

- Apply an algorithm from multiple starting points.
  - How to choose starting point?
  - How to avoid shooting in the dark?
- Allow the algorithm to proceed upwards: simulated annealing
  - Climb the mountain to find another valley.
  - How to decide when it is time to climb or to go down?
- Change the structure of the neighborhood: variable neighborhood search
  - How to choose the neighborhood structures?

Meta-heuristics

- Methods designed to escape from local optima are sometimes called "meta-heuristics".
- Plenty of variants are available in the literature.
- In general, success depends on exploiting well the properties of the problem at hand.
- VNS is one of the simplest to code.
- Additional bio-inspired methods have also been proposed and applied: genetic algorithms, ant colony optimization, etc.