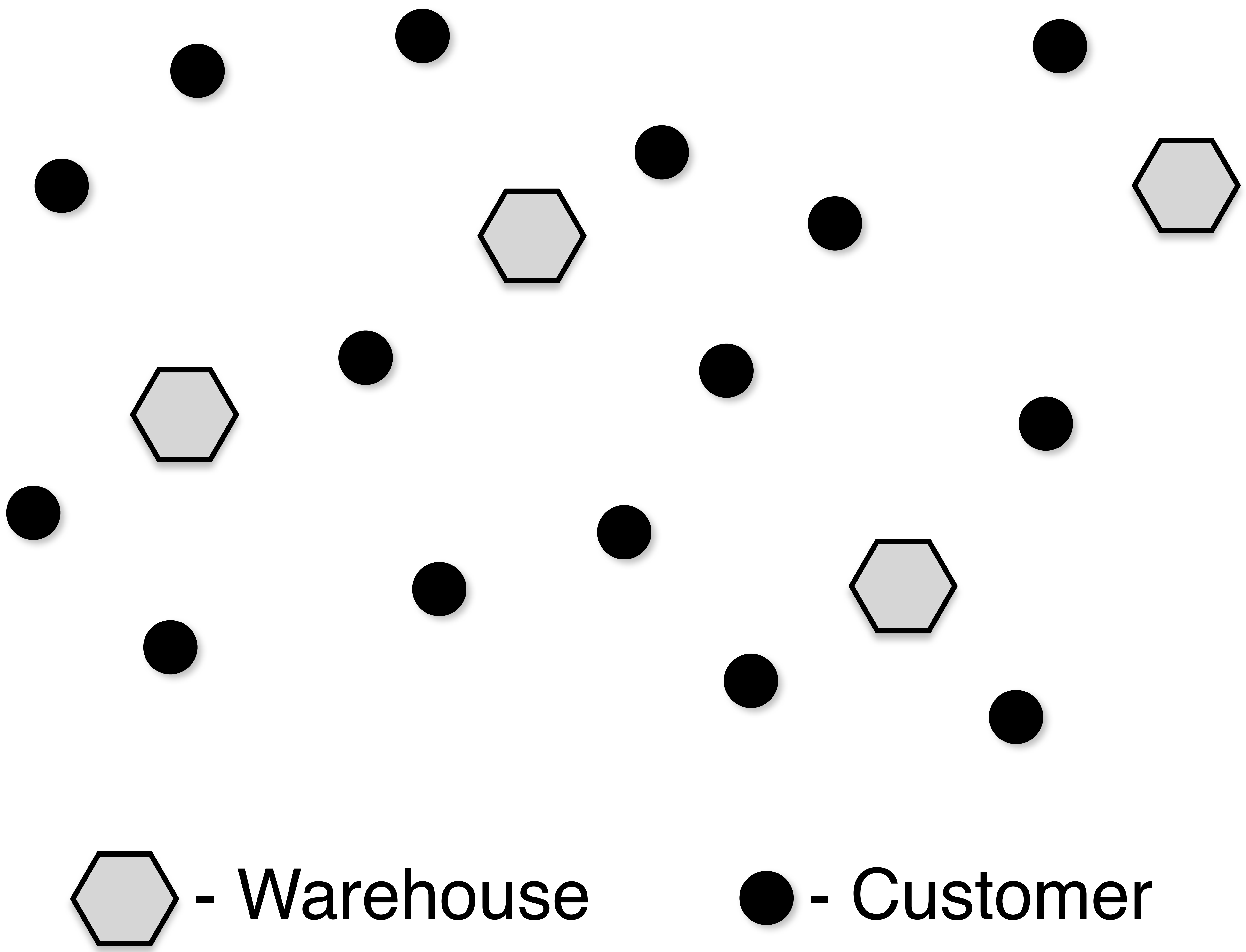
Discrete Optimization

Local Search: Part III

Goals of the Lecture

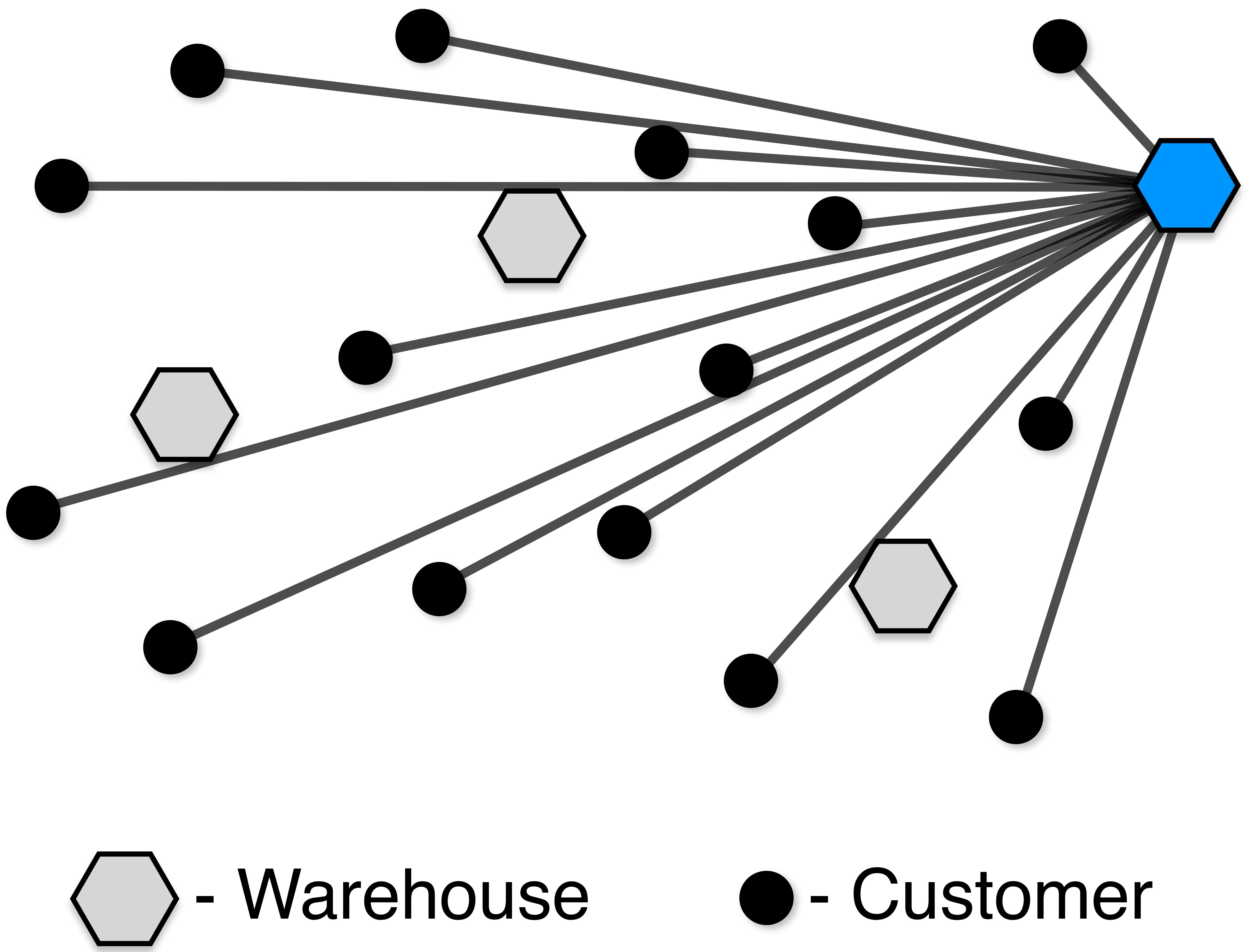
- Local search
  - optimization
  - warehouse location
  - traveling salesman problem

Warehouse Location

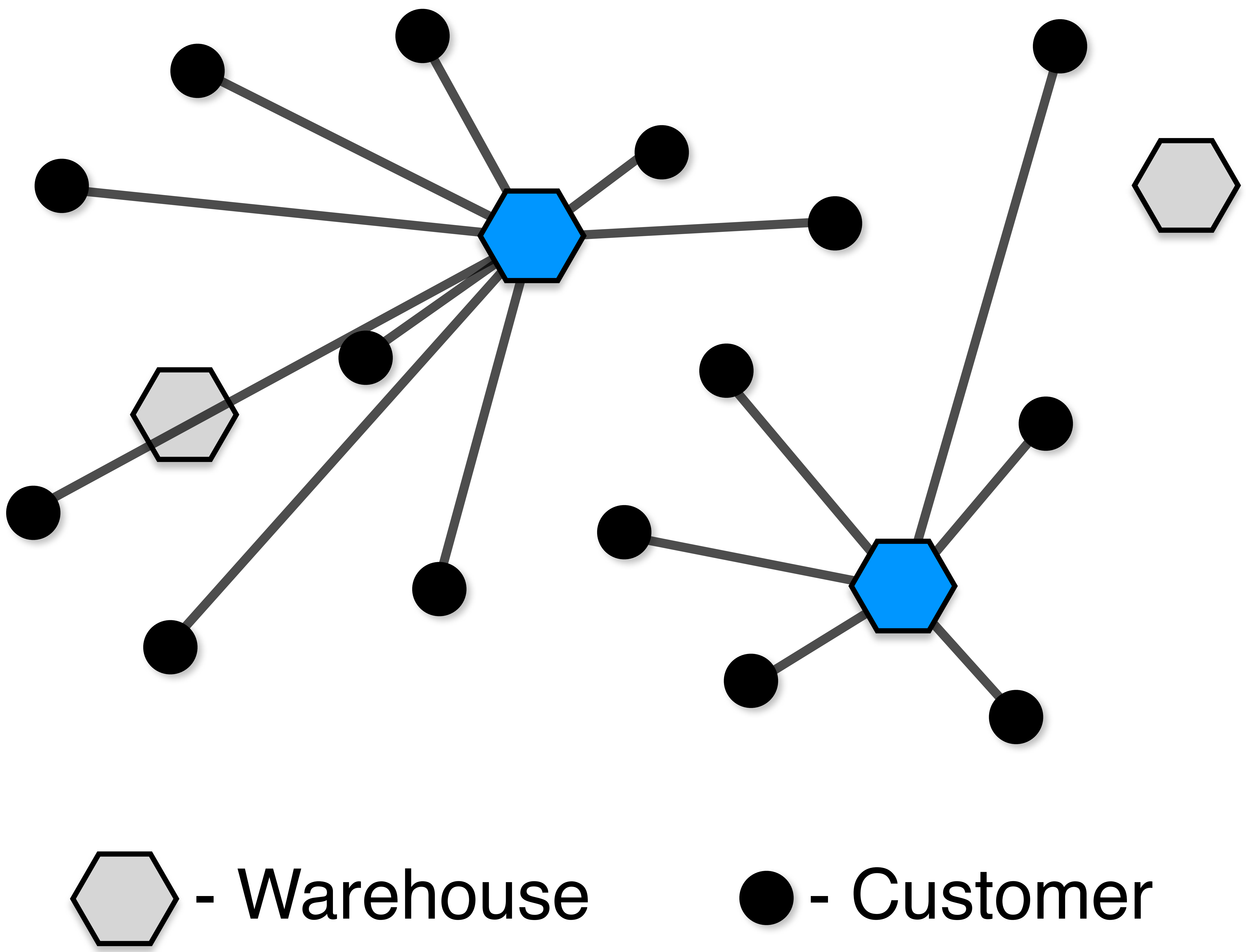


`min warehouse setup cost + transport cost`

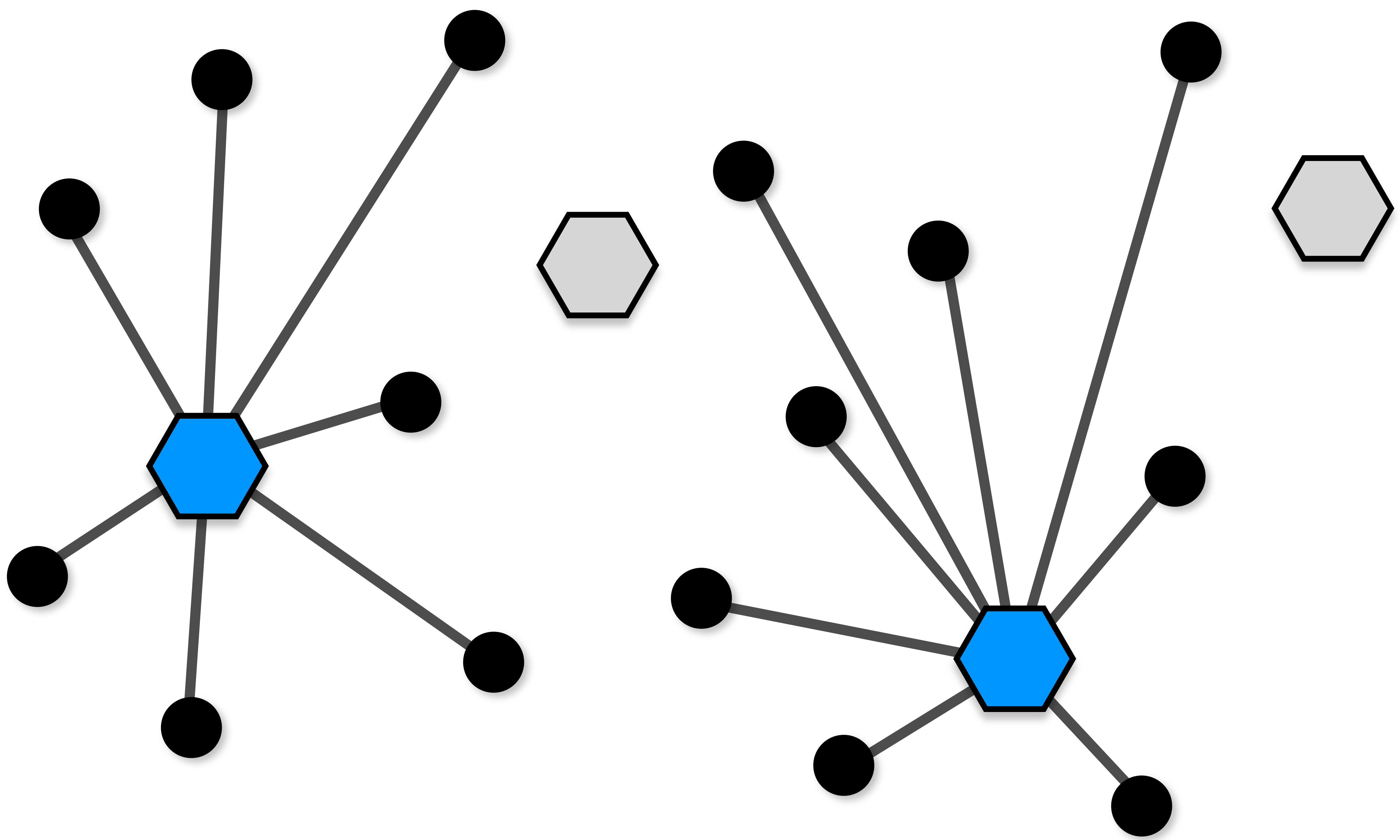
Warehouse Location



Warehouse Location



Warehouse Location



Warehouse Location

- Given
  - a set of warehouses W, each warehouse with a fixed cost f_w
  - a set of customers C
  - a transportation cost $t_{w,c}$ from warehouse w to customer c
- Find which warehouses to open to minimize the fixed and transportation costs

Warehouse Location

- What are the decision variables?
  - o_w: whether warehouse w is open (0/1)
  - a[c]: the warehouse assigned to customer c

Warehouse Location

- What are the decision variables?
  - o_w: whether warehouse w is open (0/1)
  - a[c]: the warehouse assigned to customer c
- What are the constraints?

Warehouse Location

- What are the decision variables?
  - o_w: whether warehouse w is open (0/1)
  - a[c]: the warehouse assigned to customer c
- What are the constraints?
  - no constraints

Warehouse Location

- What are the decision variables?
  - o_w: whether warehouse w is open (0/1)
  - a[c]: the warehouse assigned to customer c
- What are the constraints?
  - no constraints 🙂
- What is the objective?

Warehouse Location

- What are the decision variables?
  - o_w: whether warehouse w is open (0/1)
  - a[c]: the warehouse assigned to customer c
- What are the constraints?
  - no constraints
- What is the objective?

$$\text{minimize} \sum_{w \in W} f_w o_w + \sum_{c \in C} t_{a[c],c}$$

Warehouse Location

- Key observation
  - once the warehouse locations have been chosen, the problem is easy
  - it suffices to assign a customer to the open warehouse minimizing its transportation cost

Warehouse Location

- Key observation
  - once the warehouse locations have been chosen, the problem is easy
  - it suffices to assign a customer to the open warehouse minimizing its transportation cost
- What is the objective?

Warehouse Location

- Key observation
  - once the warehouse locations have been chosen, the problem is easy
  - it suffices to assign a customer to the open warehouse minimizing its transportation cost
- What is the objective?

$$\text{minimize} \sum_{w \in W} f_w o_w + \sum_{c \in C} \min_{w \in W : o_w = 1} t_{w,c}$$

Warehouse Location

- Neighborhood
  - many possibilities

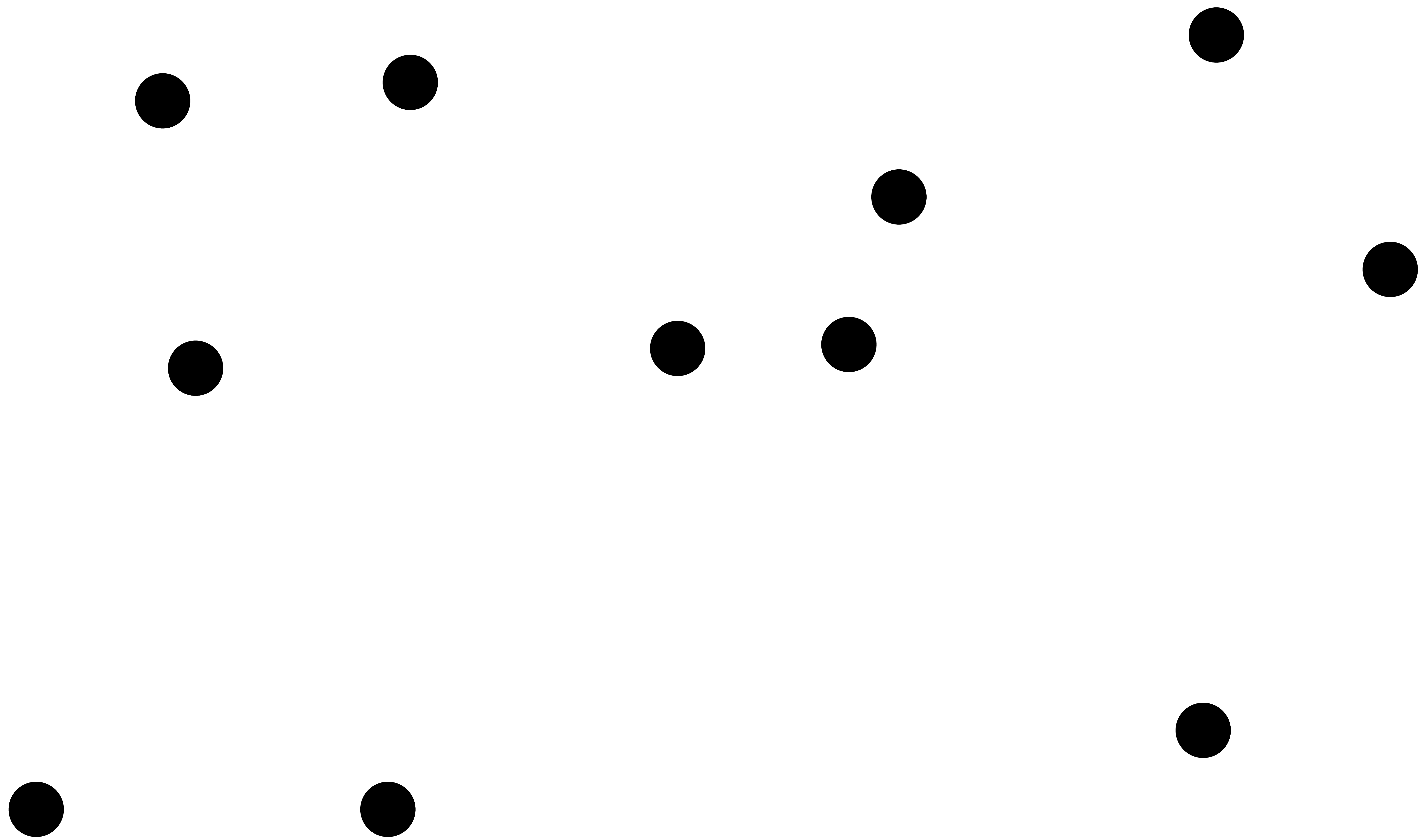
Warehouse Location

- Neighborhood
  - many possibilities
- Simplest neighborhood
  - open and close warehouses
  - that is, flip the value of o_w

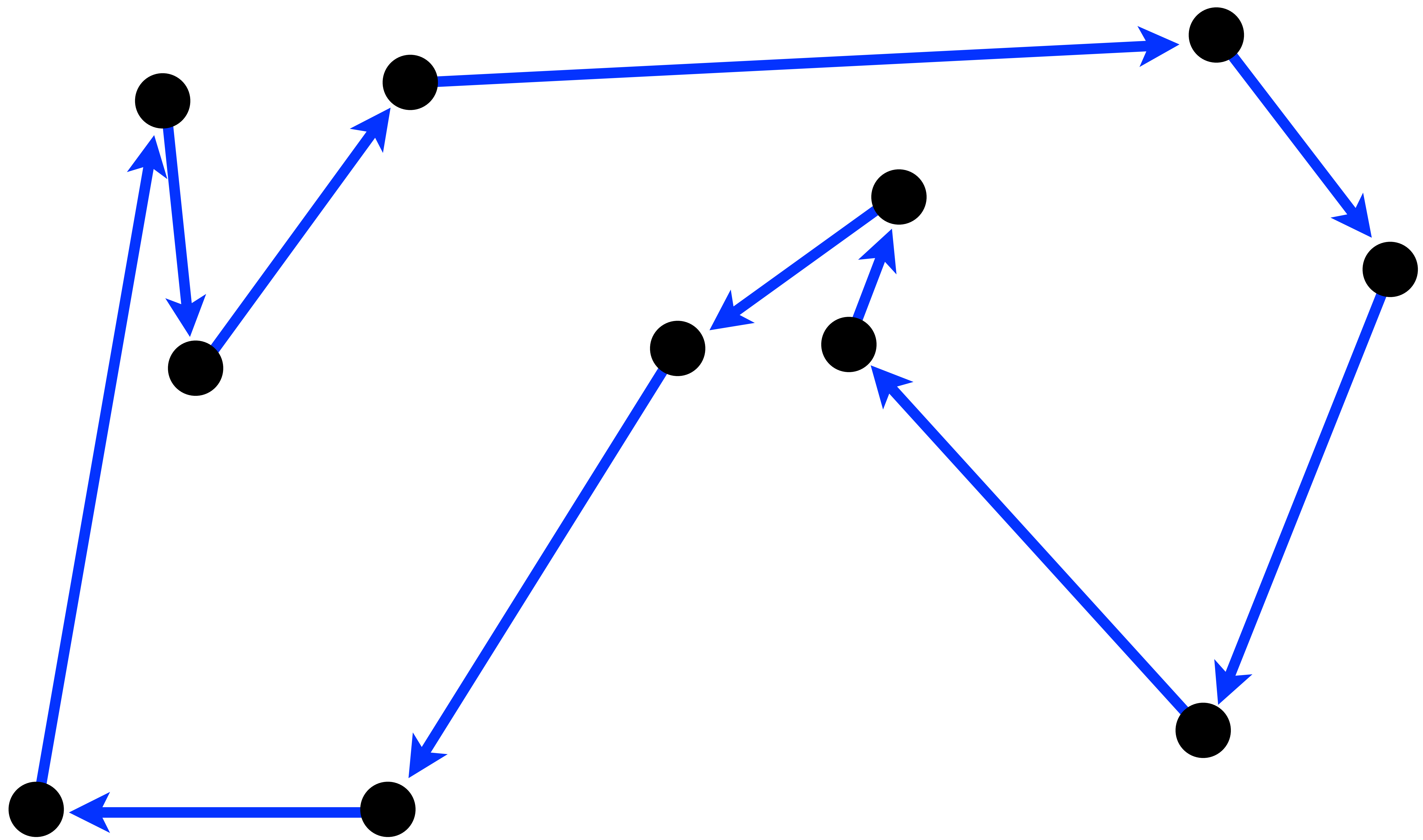
Warehouse Location

- Neighborhood
  - many possibilities
- Simplest neighborhood
  - open and close warehouses
  - that is, flip the value of o_w
- Union of neighborhoods
  - open and close a warehouse
  - swap two warehouses
    - close one and open the other

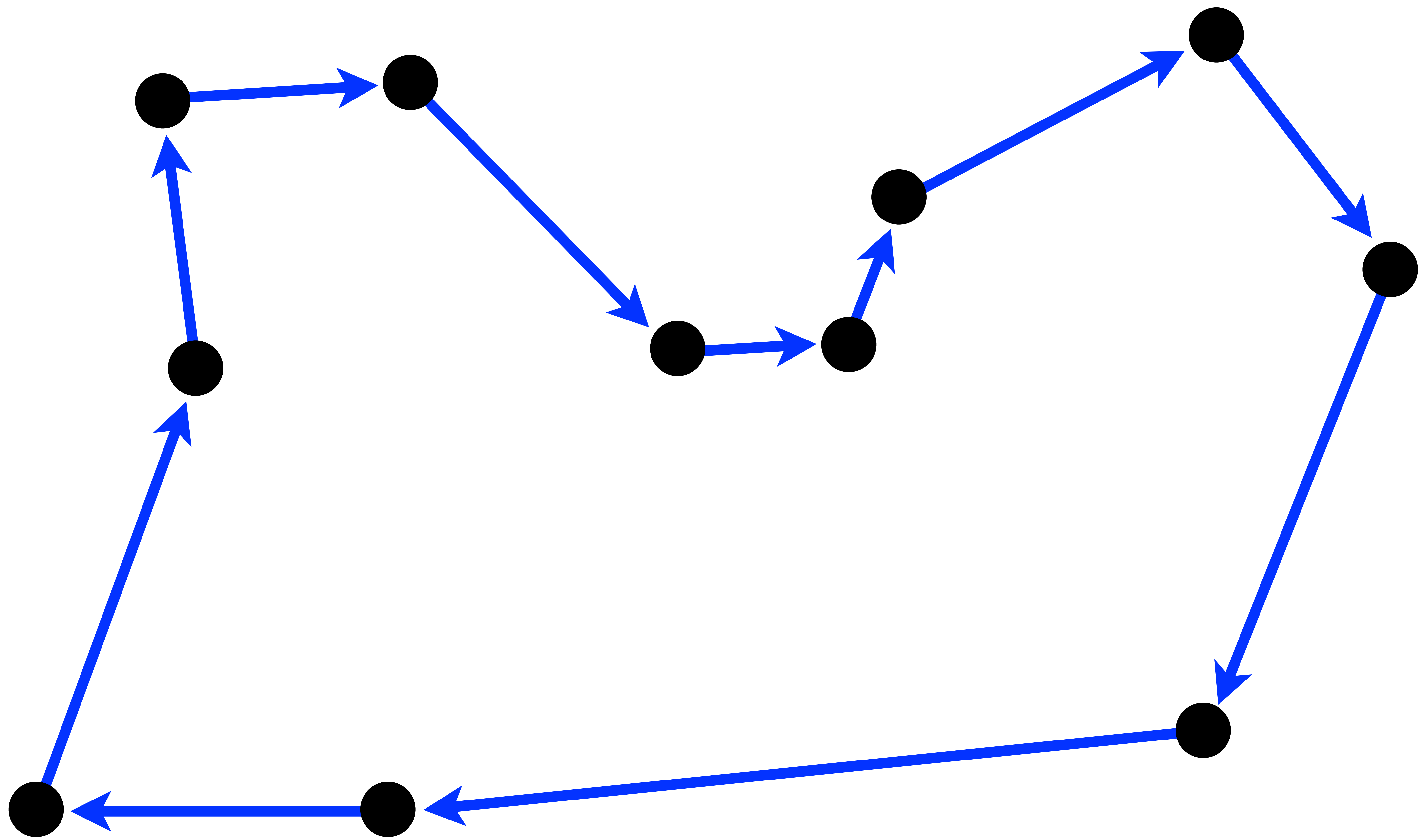
Traveling Salesman Problem



Traveling Salesman Problem



Traveling Salesman Problem



Traveling Salesman Problem

- Given
  - a set C of cities to visit
  - a symmetric distance matrix d between every two cities

Traveling Salesman Problem

- Given
  - a set C of cities to visit
  - a symmetric distance matrix d between every two cities
- Find
  - a tour of minimal cost visiting each city exactly once

Traveling Salesman Problem

- Given
  - a set C of cities to visit
  - a symmetric distance matrix d between every two cities
- Find
  - a tour of minimal cost visiting each city exactly once
- The traveling salesman problem (TSP) is probably the most studied combinatorial problem

Traveling Salesman Problem

- Decision variables
  – like in the Euler tour
  – specify where to go next for every city

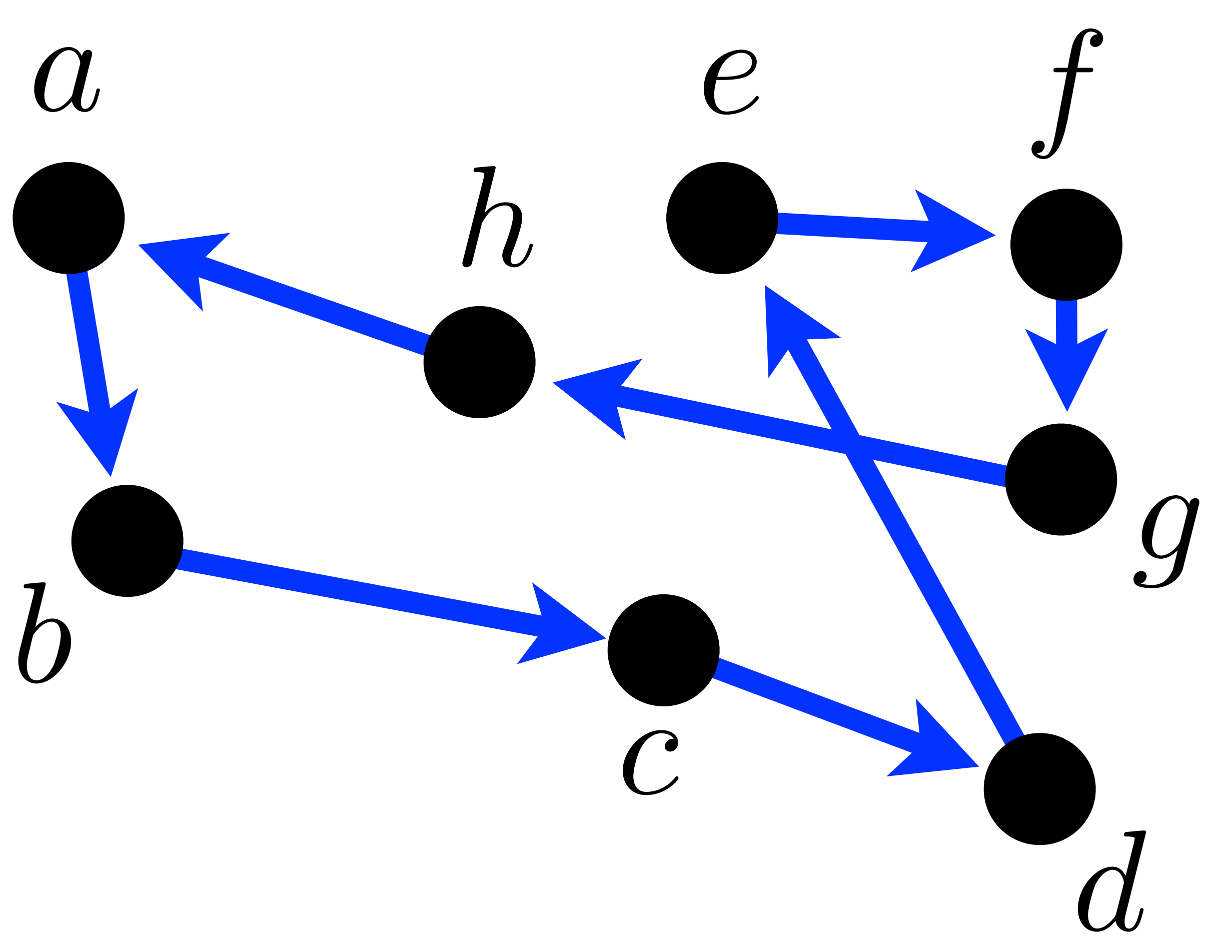
```
range Cities = 1..n;
int distance[Cities,Cities] = ...;
var{int} next[Cities] in Cities;
minimize
    sum(c in Cities) d[c,next[c]]
subject to
    circuit(next);
```

Local Search for the TSP

- 2-OPT neighborhood for the TSP
  - stay feasible, that is always maintain a tour
  - select two edges and replace them by two other edges

Local Search for the TSP

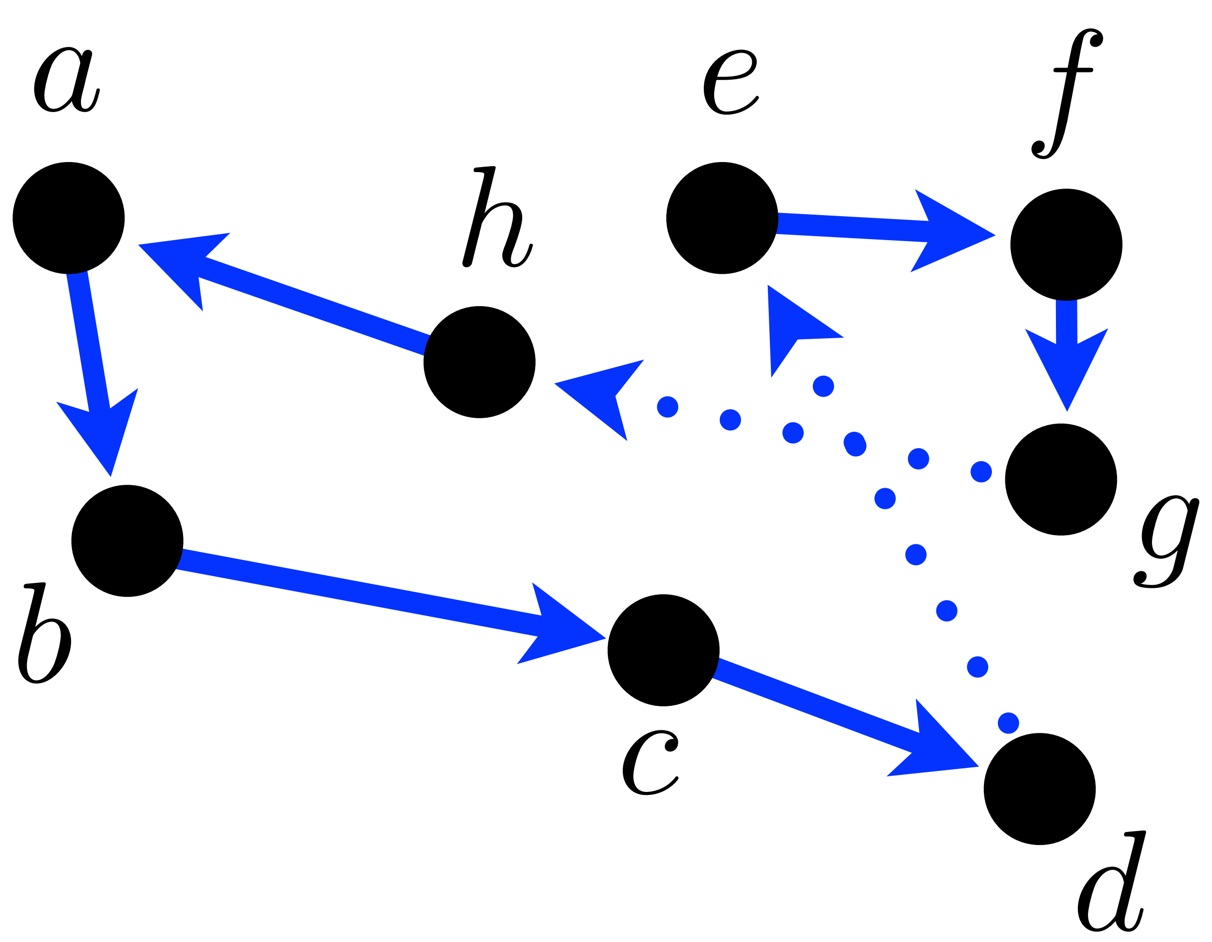
- 2-OPT neighborhood for the TSP
  - stay feasible, that is always maintain a tour
  - select two edges and replace them by two other edges



$$a \to b \to c \to d \to e \to f \to g \to h \to a$$

Local Search for the TSP

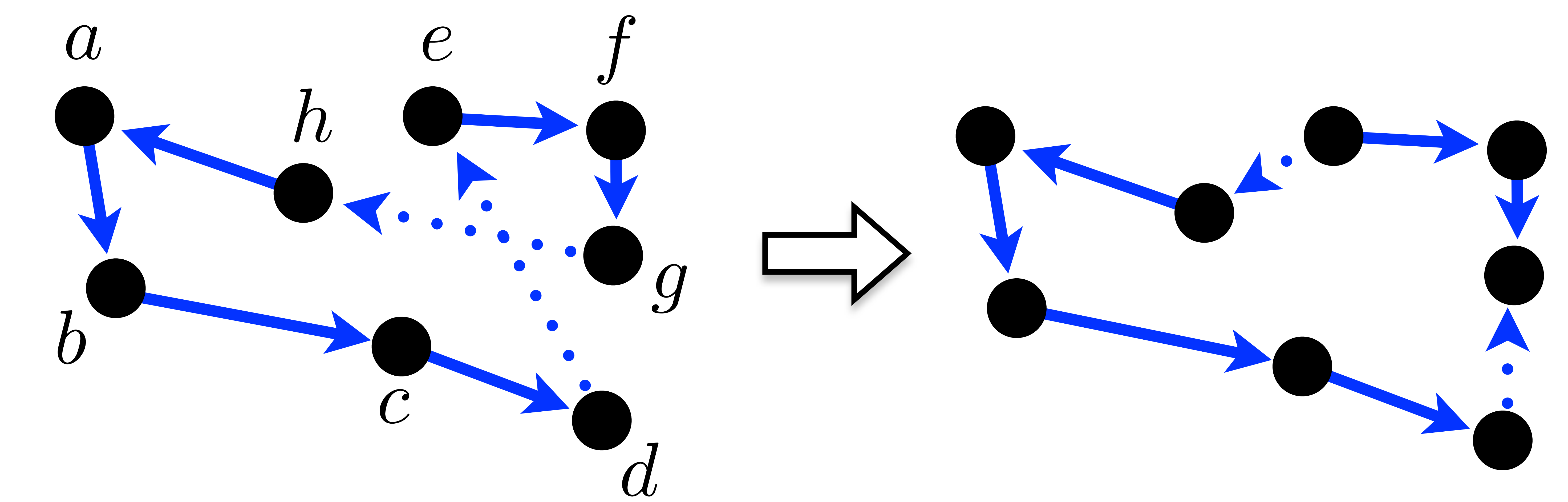
- 2-OPT neighborhood for the TSP
  - stay feasible, that is always maintain a tour
  - select two edges and replace them by two other edges



$$a \to b \to c \to d \not\to e \to f \to g \not\to h \to a$$

Local Search for the TSP

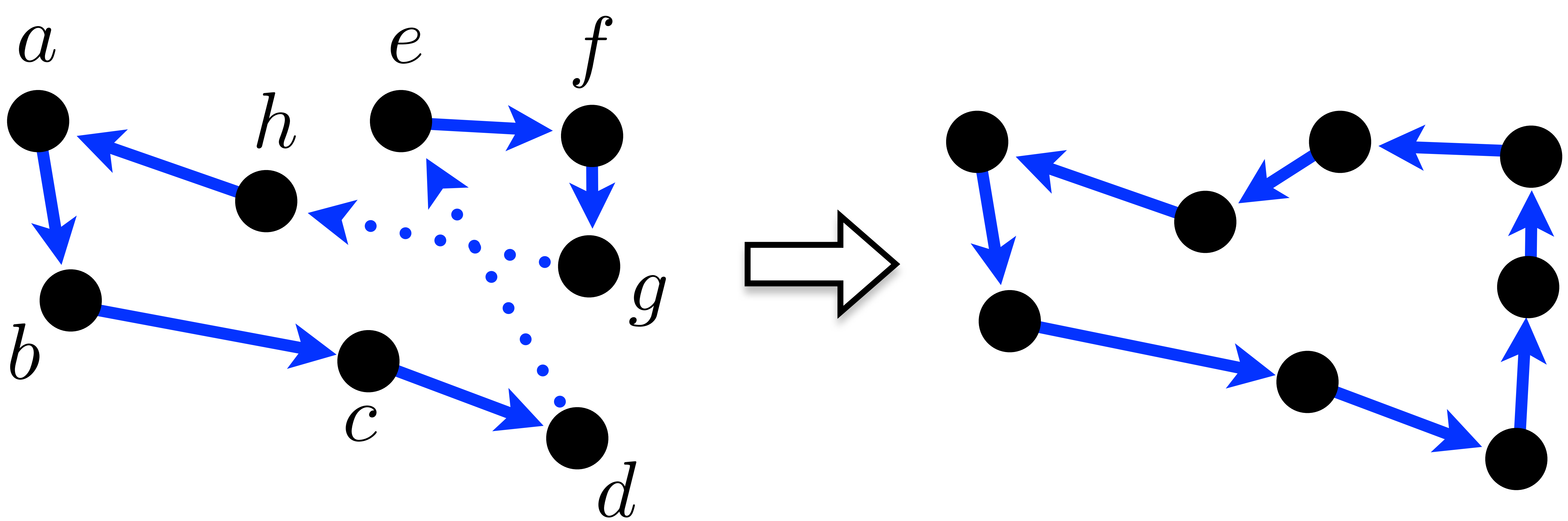
- 2-OPT neighborhood for the TSP
  - stay feasible, that is always maintain a tour
  - select two edges and replace them by two other edges



$a \to b \to c \to d \not\to e \to f \to g \not\to h \to a$ $\quad$ $a \to b \to c \to d \not\to g \leftarrow f \leftarrow e \not\to h \to a$

Local Search for the TSP

- 2-OPT neighborhood for the TSP
  - stay feasible, that is always maintain a tour
  - select two edges and replace them by two other edges



$a \to b \to c \to d \not\to e \to f \to g \not\to h \to a$ $\quad$ $a \to b \to c \to d \to g \to f \to e \to h \to a$

Local Search for the TSP

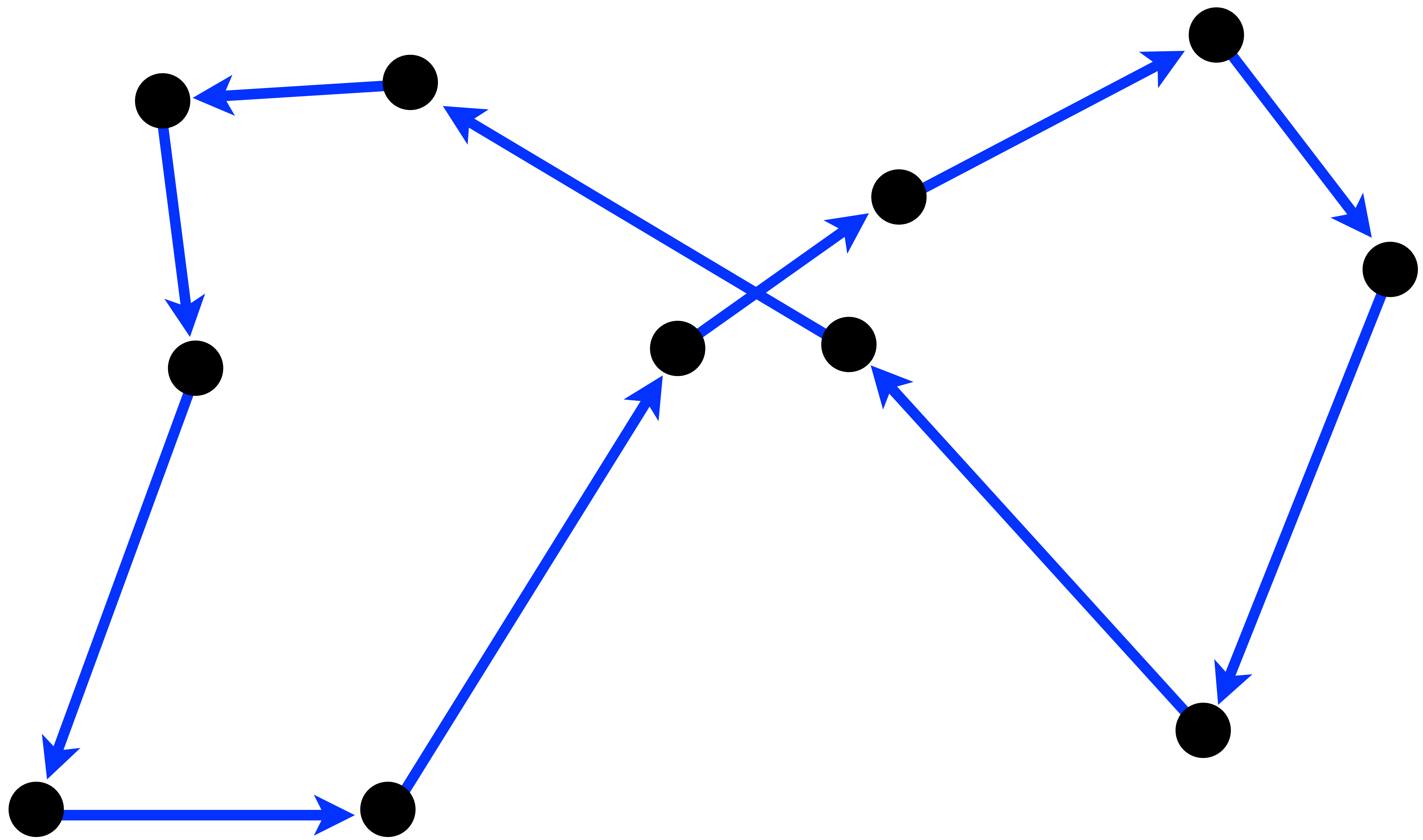
- 2-OPT
  - the neighborhood is the set of all tours that can be reached by swapping two edges
  - select two edges and replace them by two other edges

Local Search for the TSP

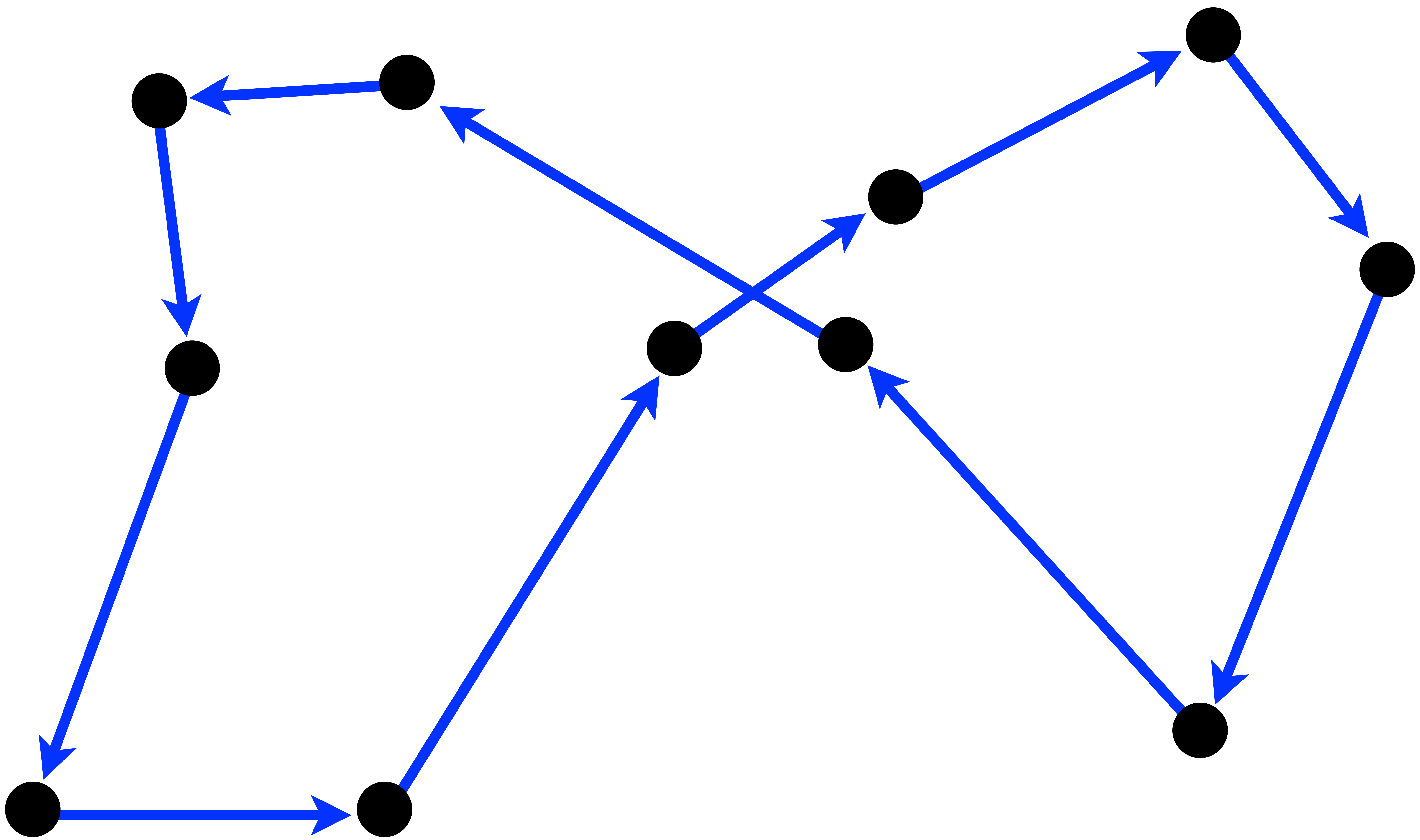
- 2-OPT
  - the neighborhood is the set of all tours that can be reached by swapping two edges
  - select two edges and replace them by two other edges
- 3-OPT
  - the neighborhood is the set of all tours that can be reached by swapping three edges

better than 2-OPT in quality
expensive in computing

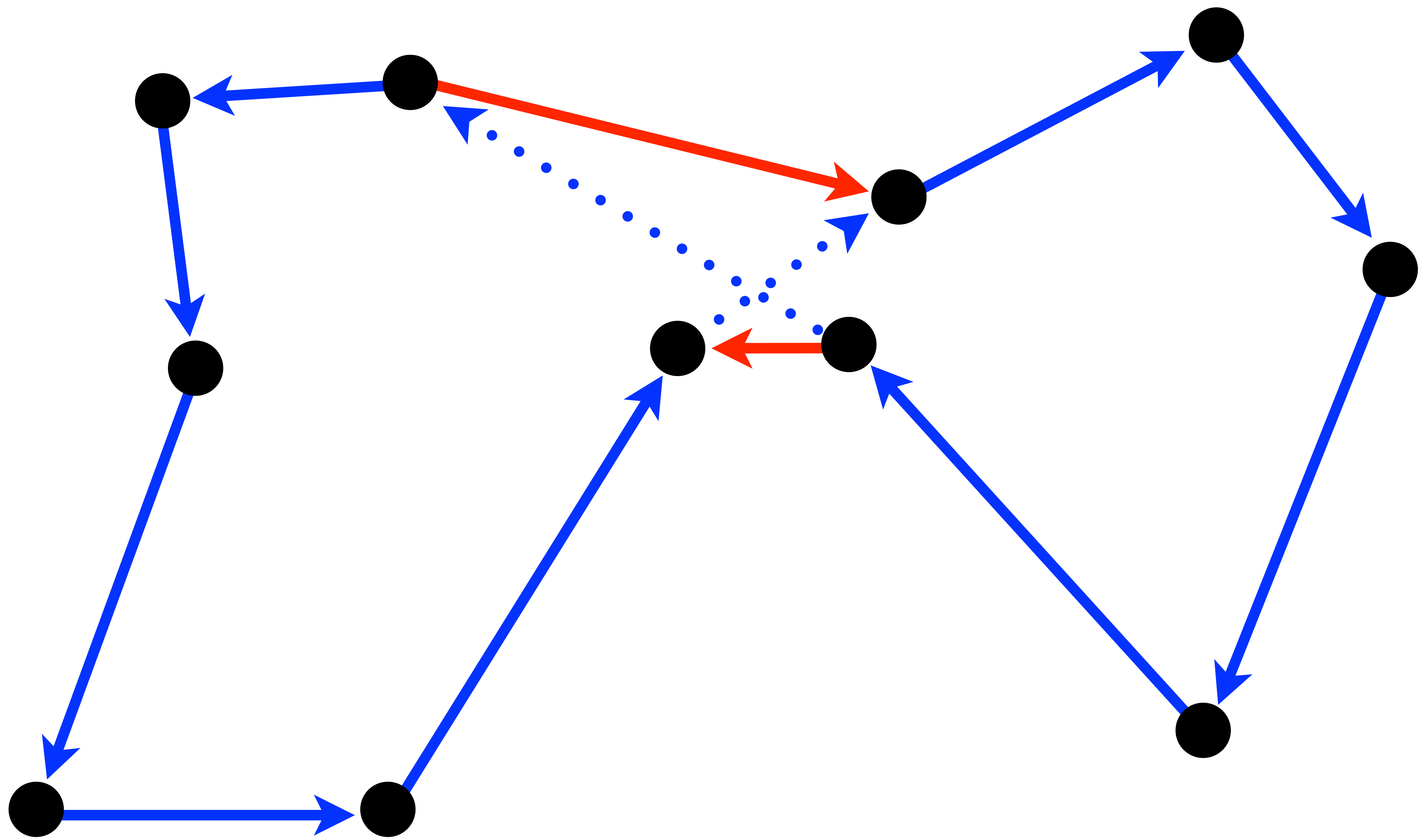
2-OPT



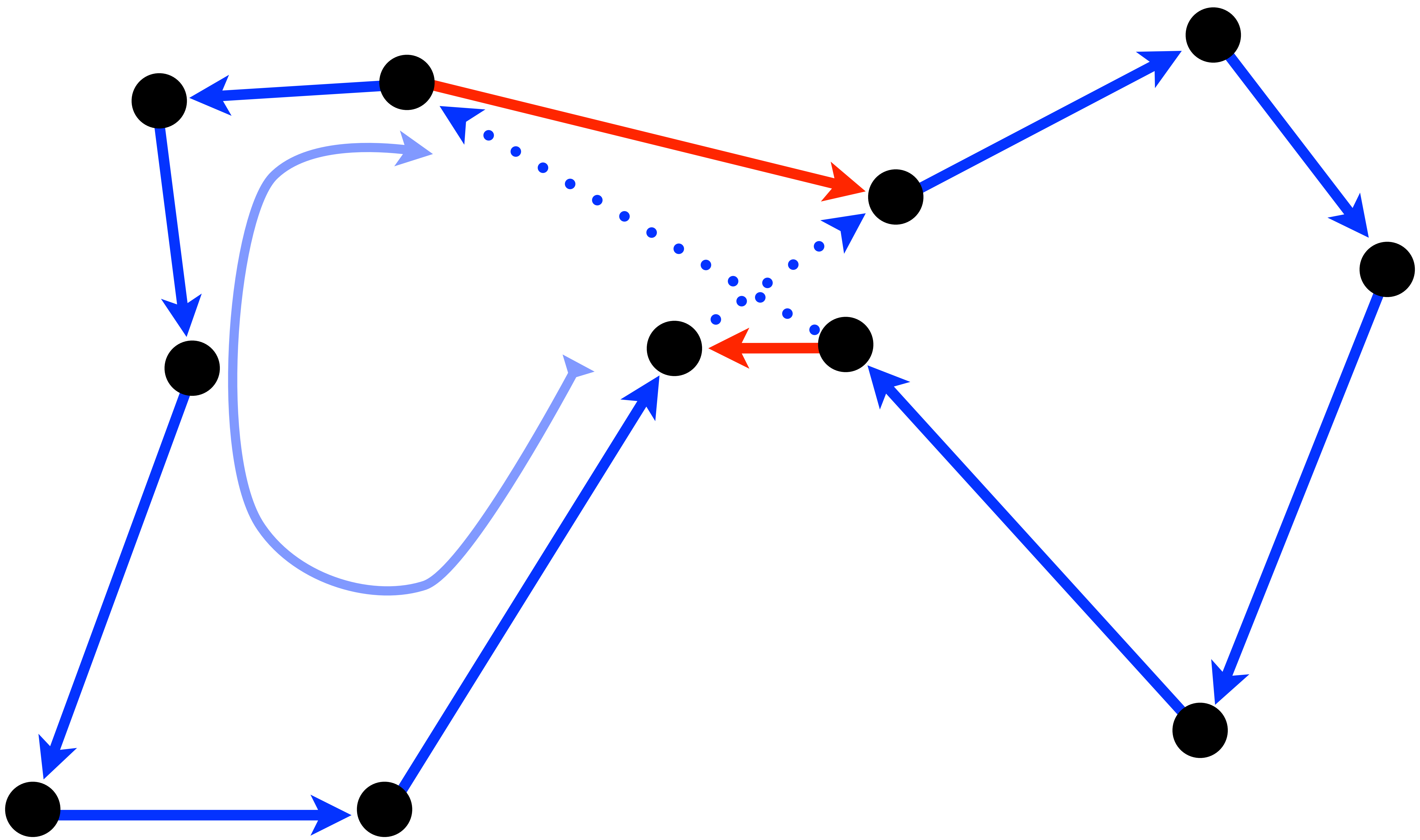
2-OPT



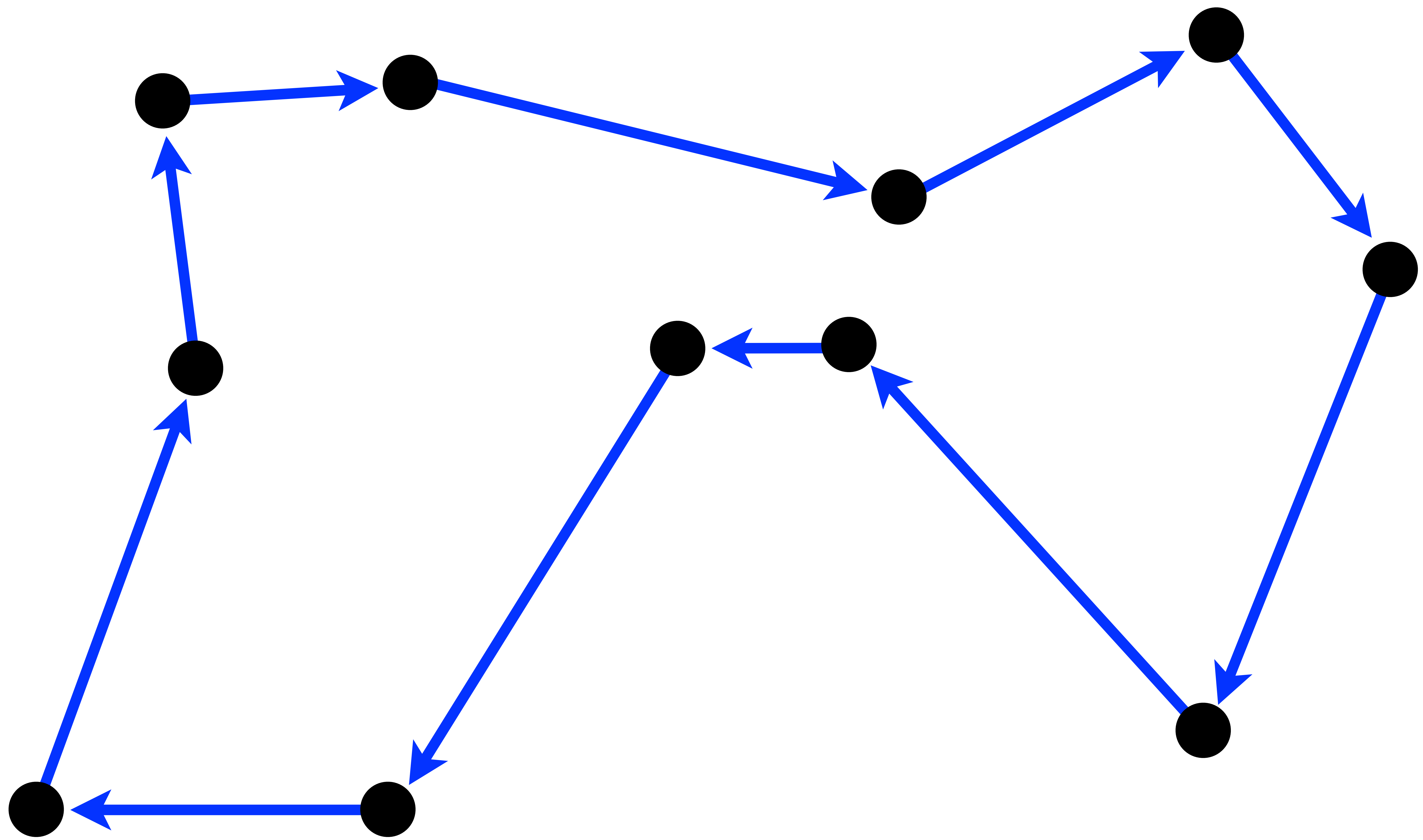
2-OPT



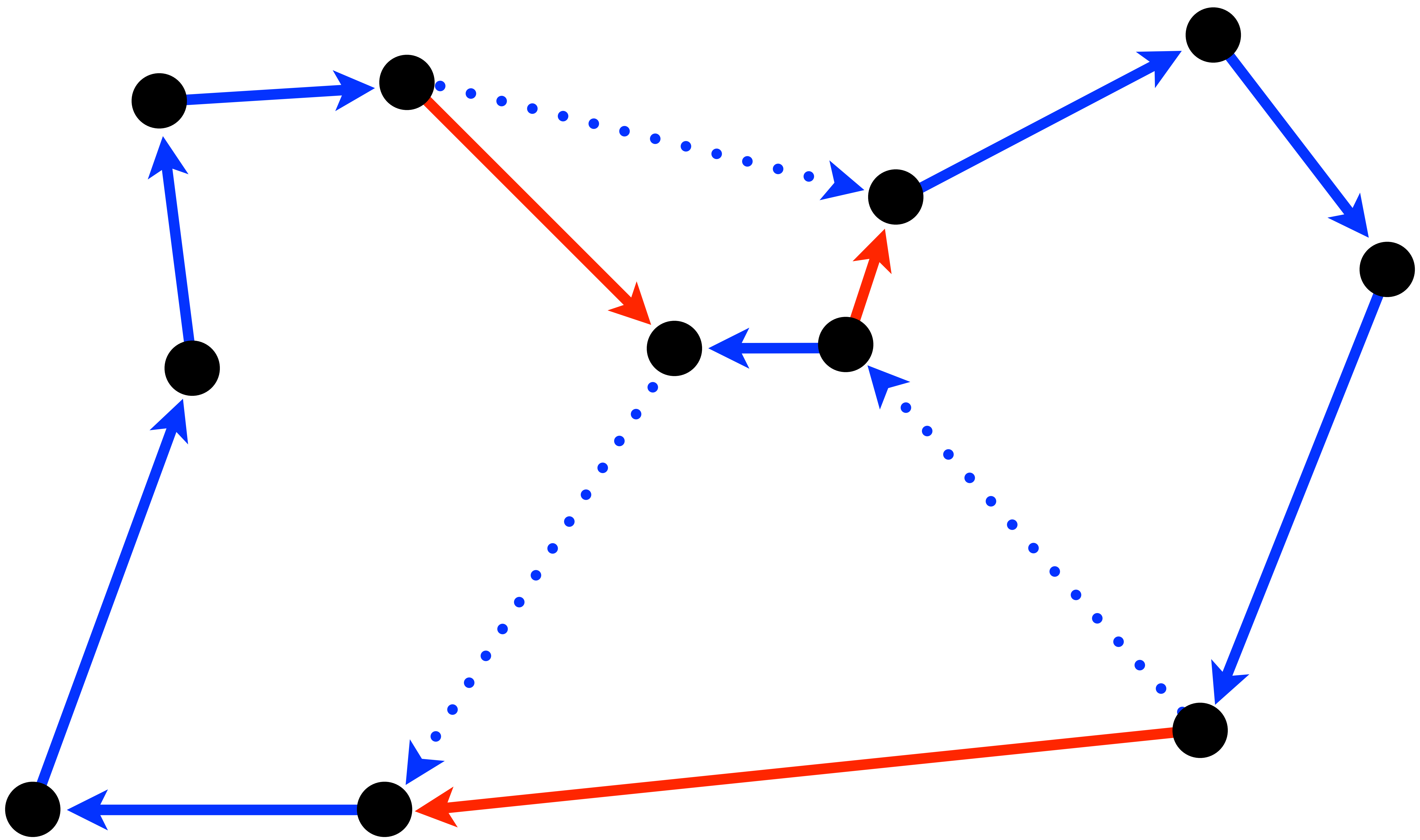
2-OPT



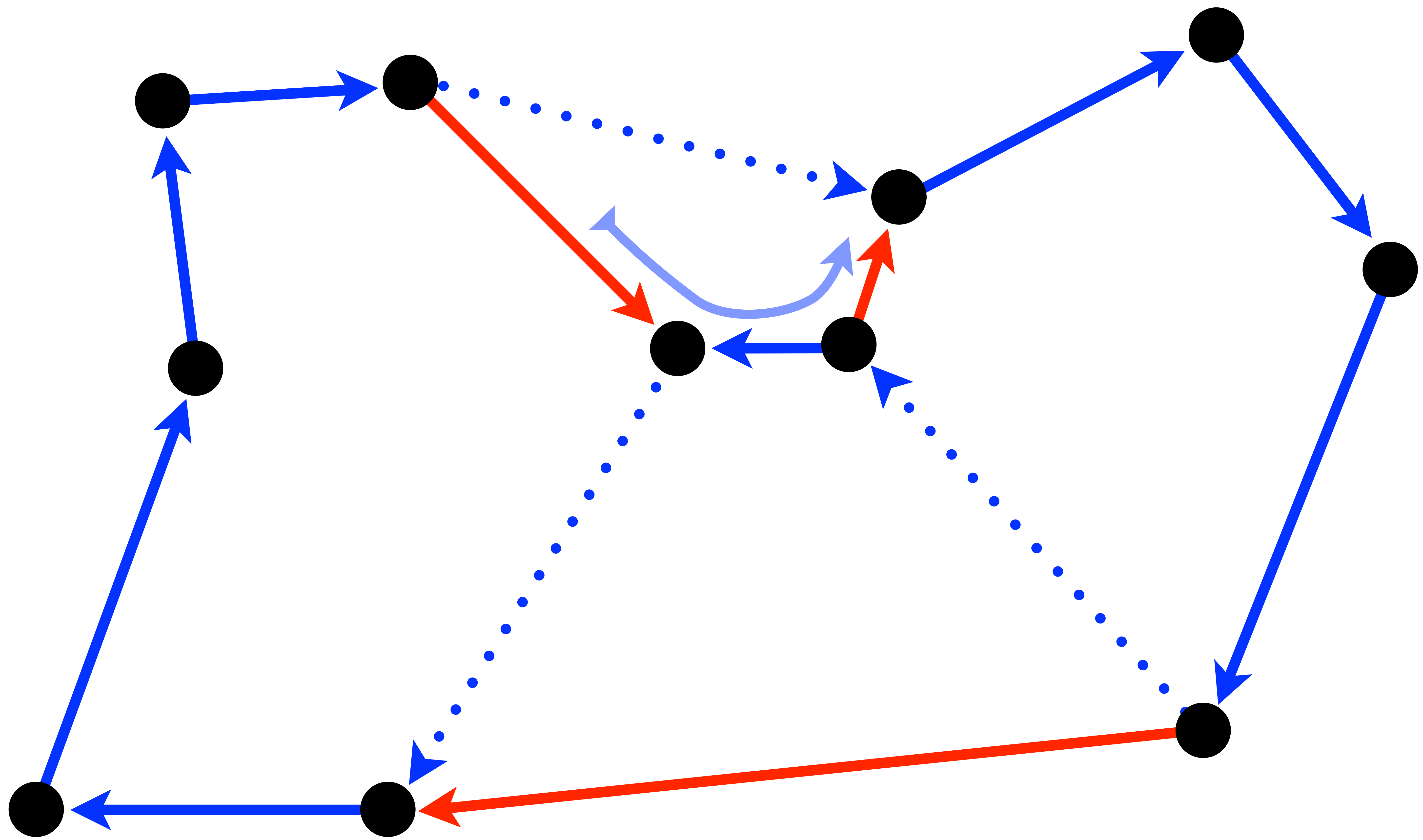
3-OPT



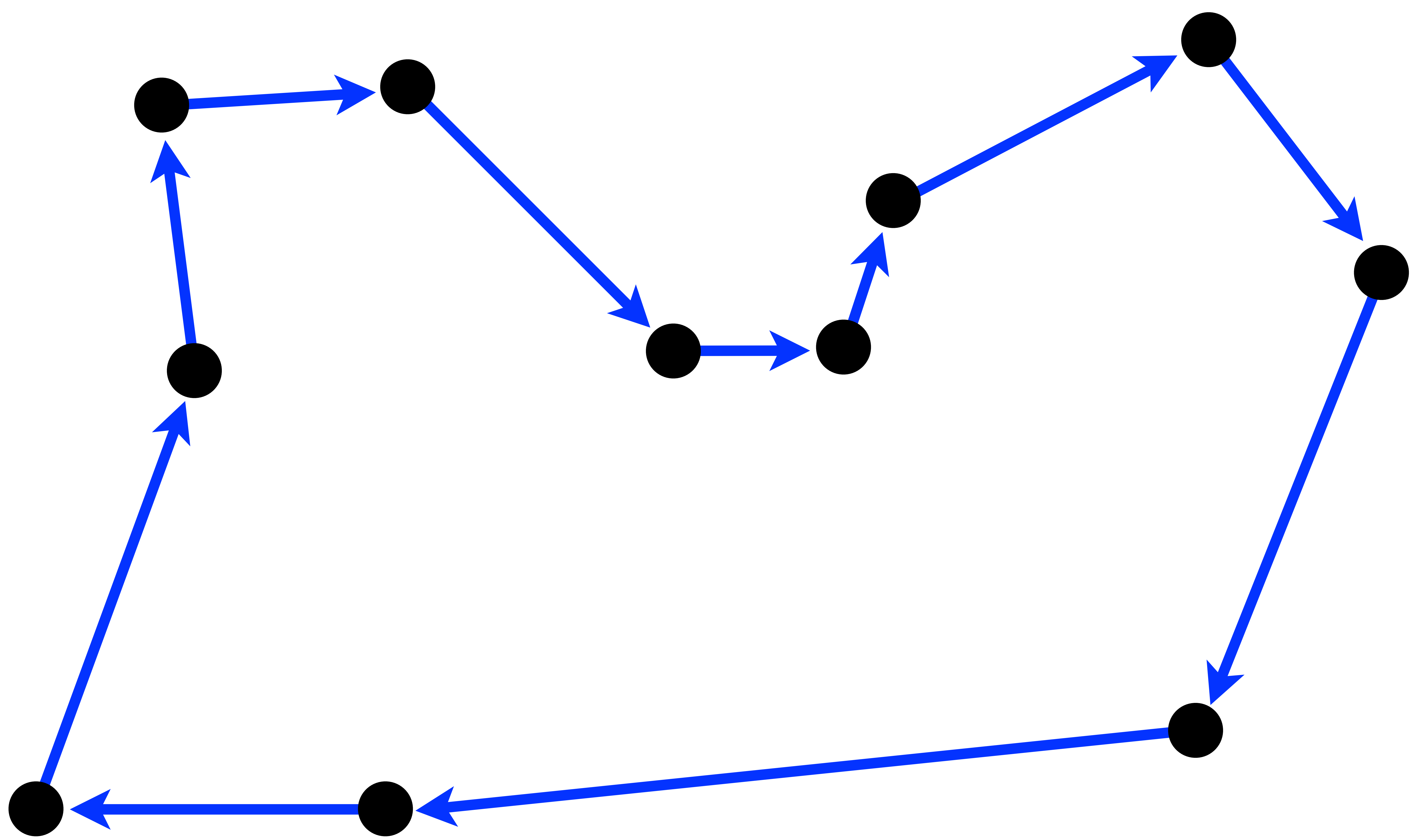
3-OPT



3-OPT



3-OPT



Local Search for the TSP

- 2-OPT
  - the neighborhood is the set of all tours that can be reached by swapping two edges
  - select two edges and replace them by two other edges

Local Search for the TSP

- 2-OPT
  - the neighborhood is the set of all tours that can be reached by swapping two edges
  - select two edges and replace them by two other edges
- 3-OPT
  - the neighborhood is the set of all tours that can be reached by swapping three edges
  - much better than 2-OPT in quality but more expensive

4-OPT

- 2-OPT
- 3-OPT
  - the neighborhood is the set of all tours that can be reached by swapping three edges
  - much better than 2-OPT in quality but more expensive

4-OPT

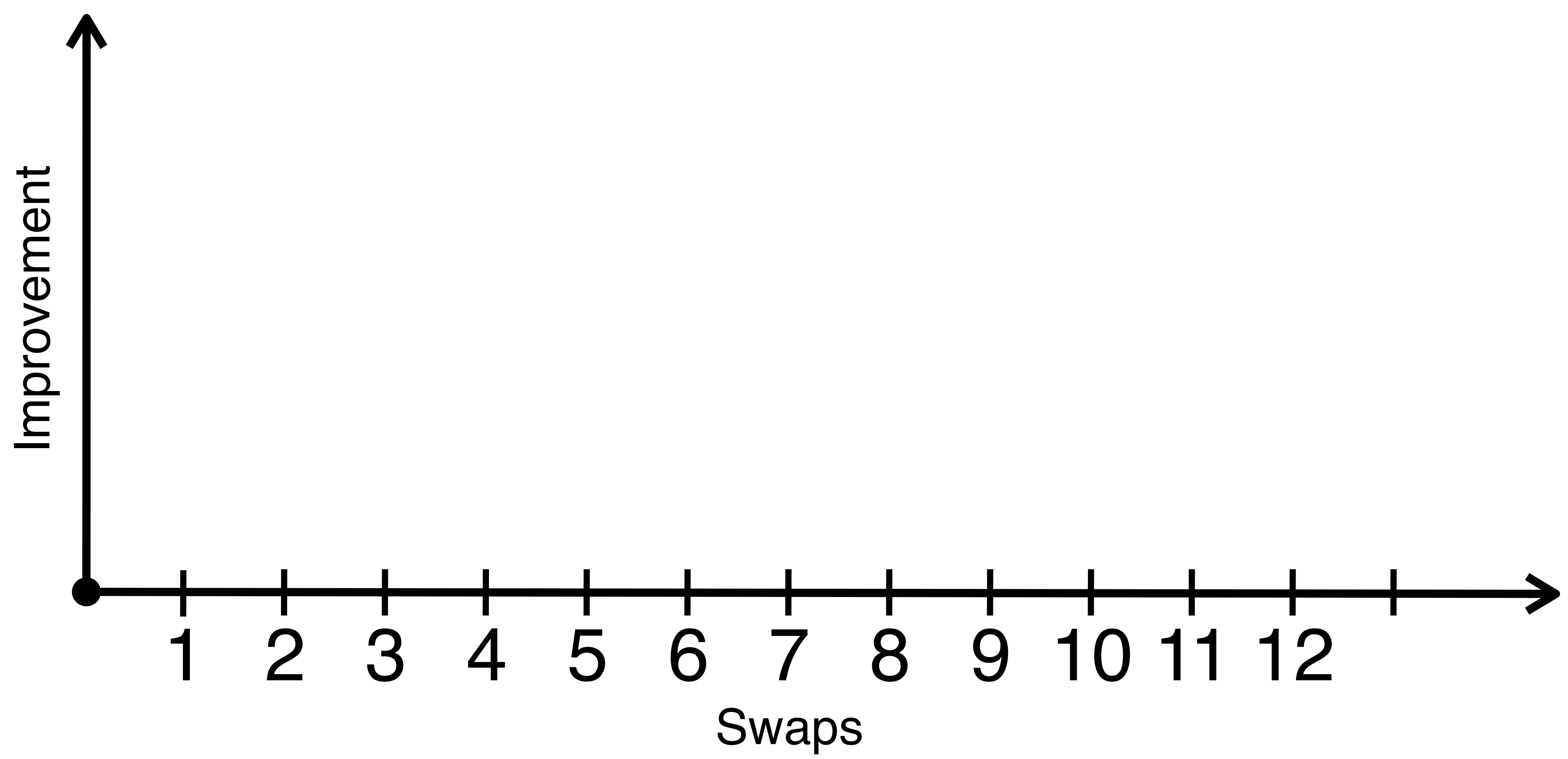
- 2-OPT
- 3-OPT
  - the neighborhood is the set of all tours that can be reached by swapping three edges
  - much better than 2-OPT in quality but more expensive
- 4-OPT
  - often marginally better but much more expensive

K-OPT

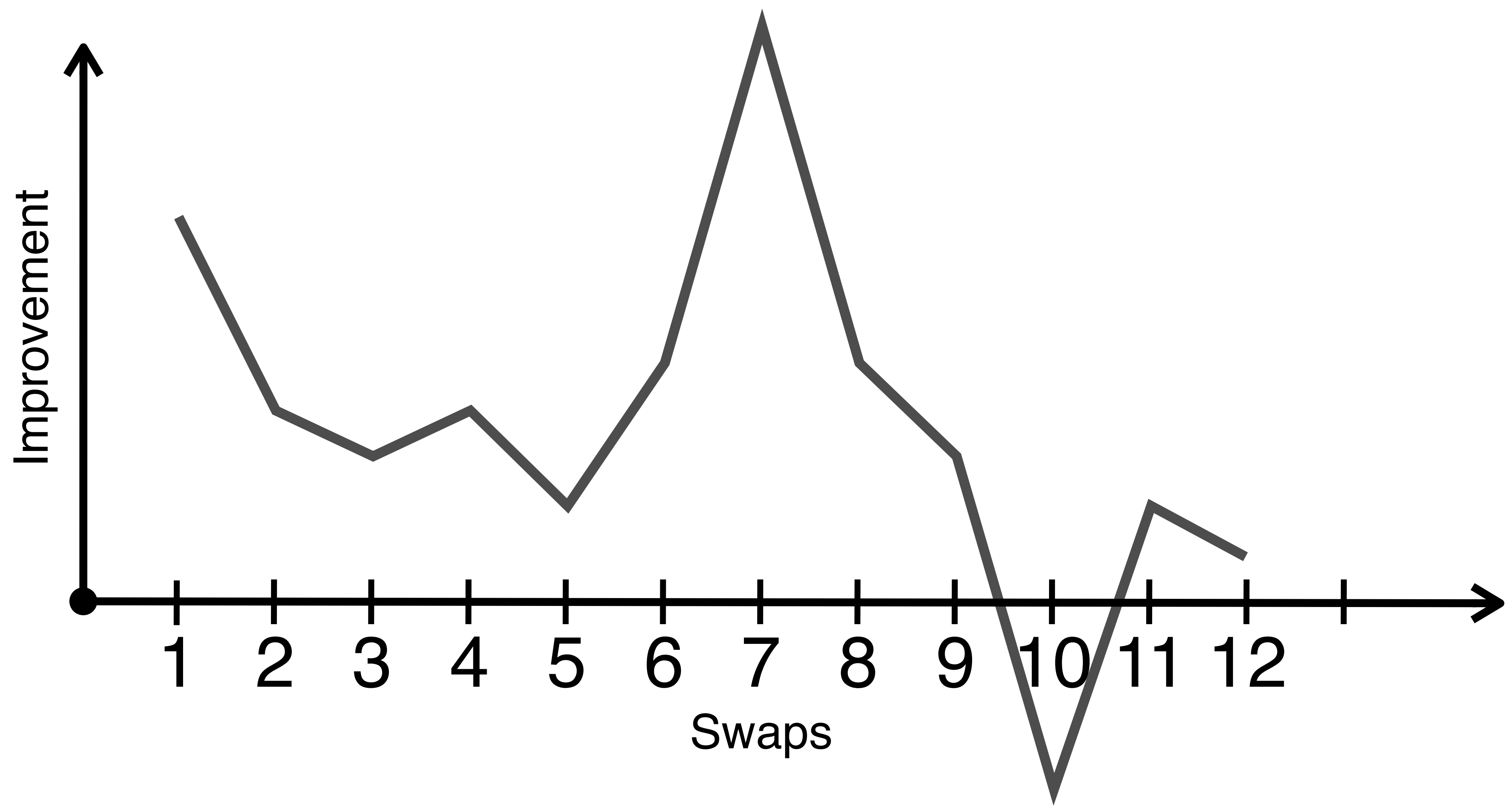
- K-OPT
  - replace the notion of one favorable swaps by a search of a favorable sequence of swaps
  - do not search for the entire set of sequences but build one incrementally

K-OPT

- K-OPT
  - replace the notion of one favorable swaps by a search of a favorable sequence of swaps
  - do not search for the entire set of sequences but build one incrementally

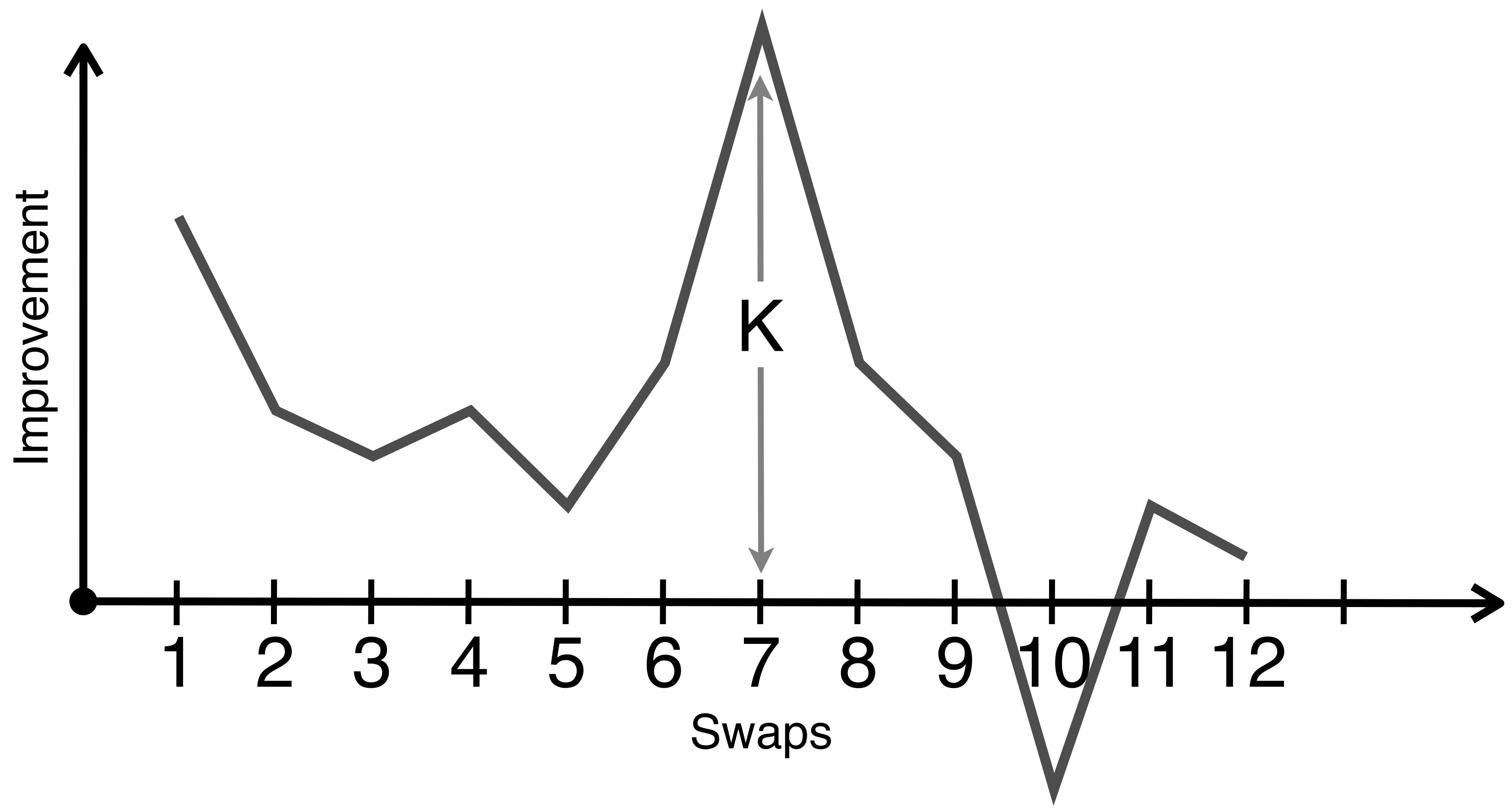


- K-OPT
  - replace the notion of one favorable swaps by a search of a favorable sequence of swaps
  - do not search for the entire set of sequences but build one incrementally



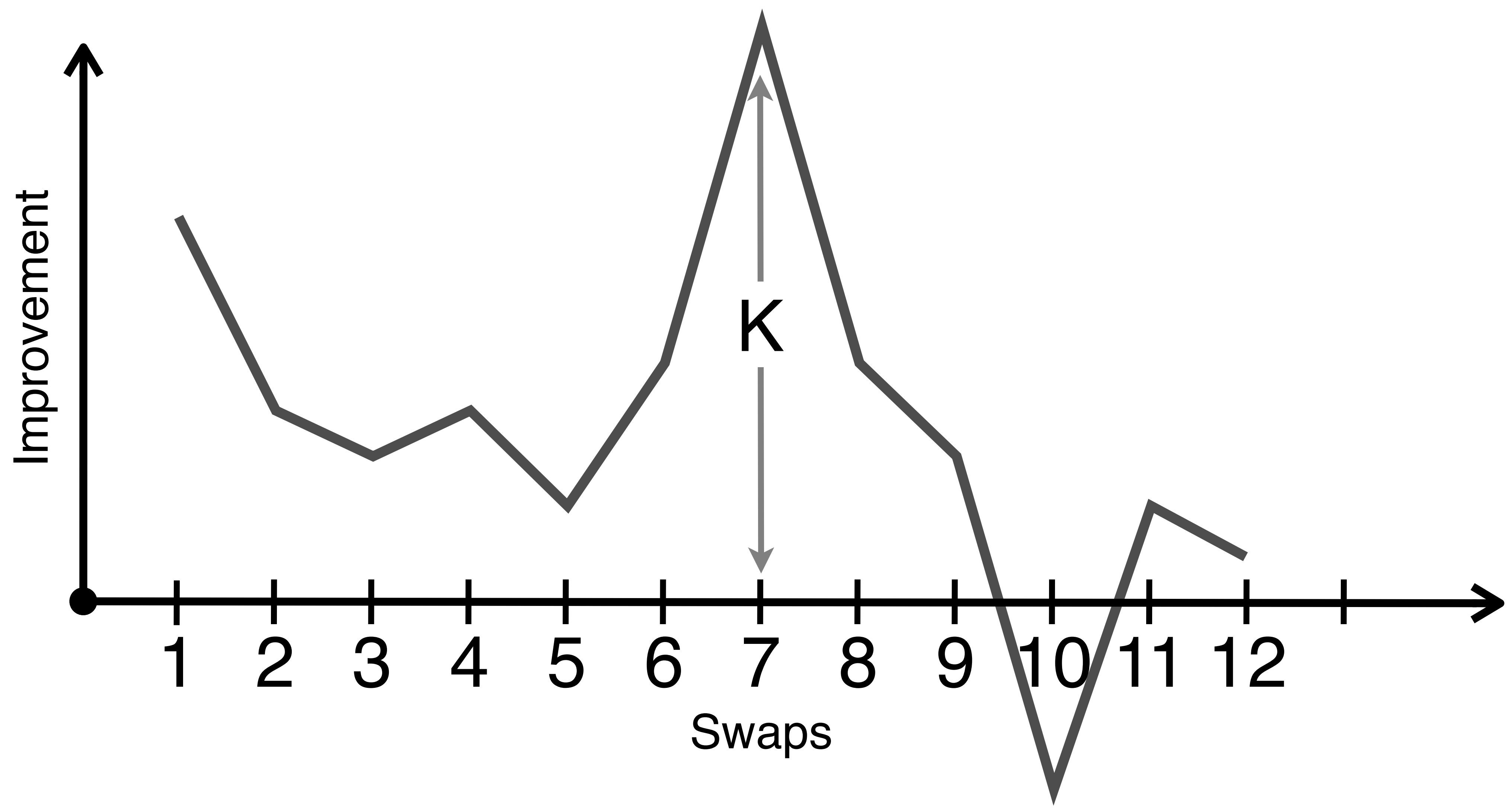
K-OPT

- K-OPT
  - replace the notion of one favorable swaps by a search of a favorable sequence of swaps
  - do not search for the entire set of sequences but build one incrementally



K-OPT

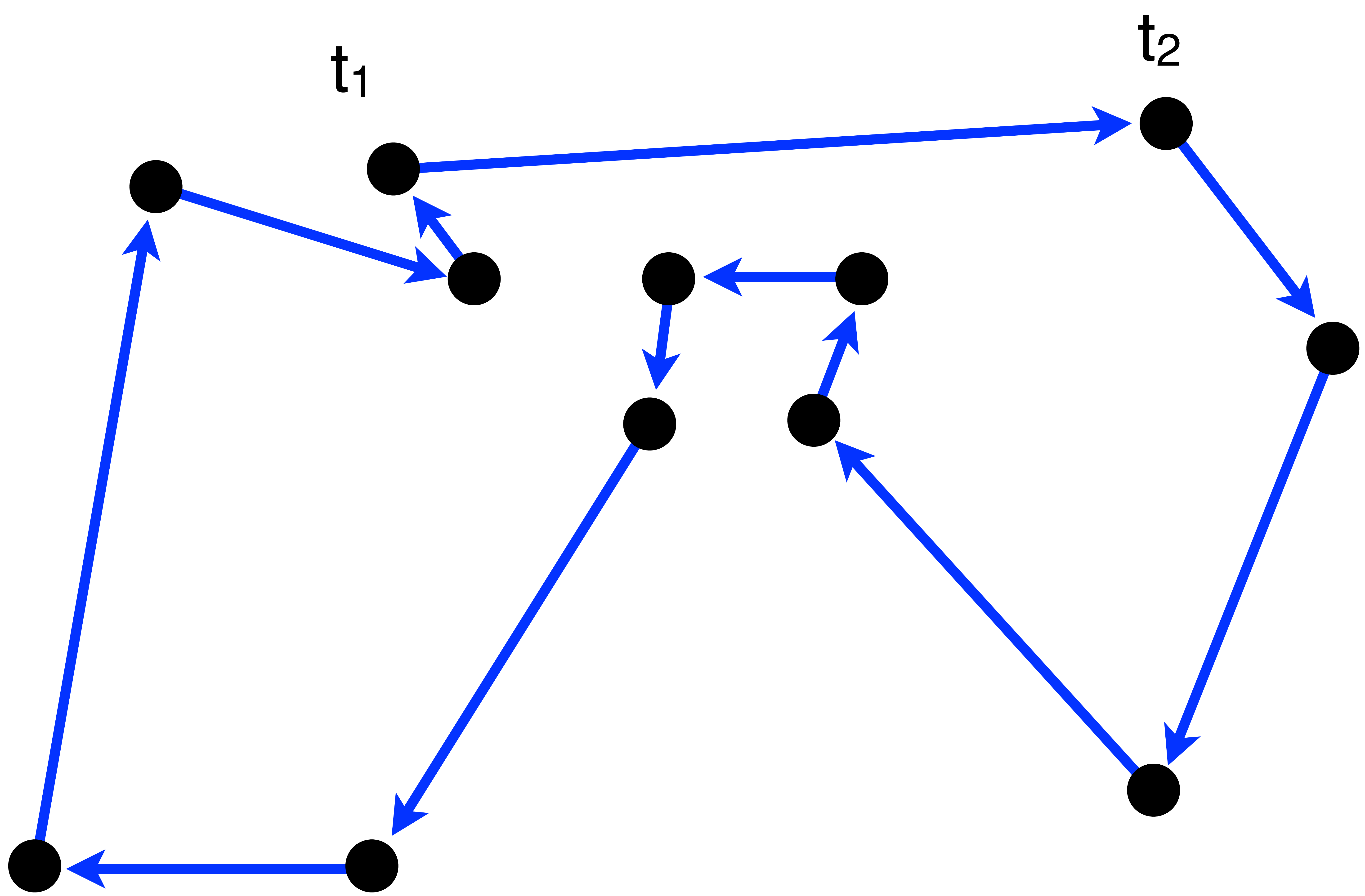
- K-OPT
  - find a good k dynamically at a fraction of the cost
  - explore a sequence of swaps of increasing sizes



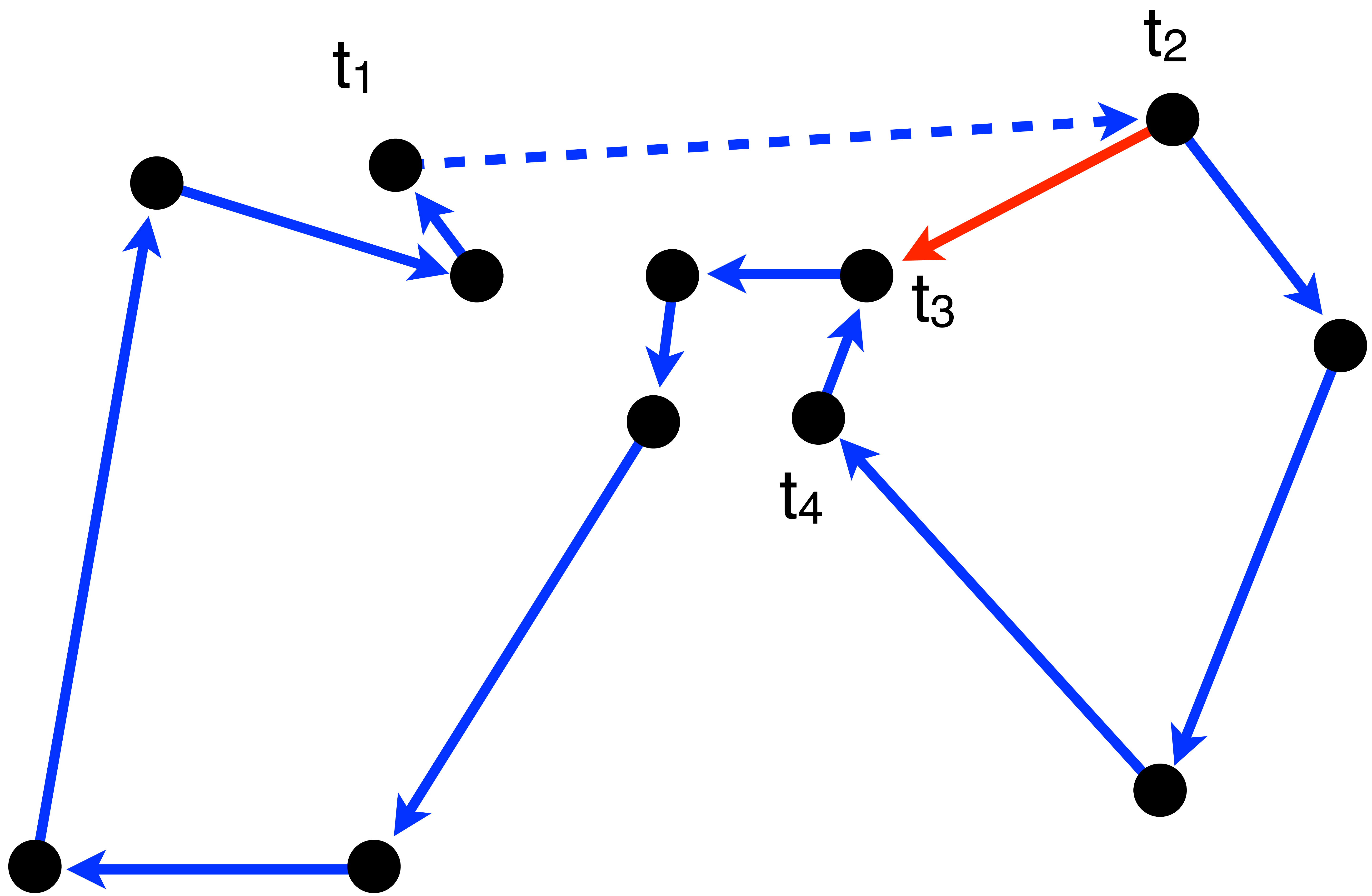
K-OPT

- K-OPT
  - choose a vertex t_1 and its edge $x_1 = (t_1, t_2)$
  - choose an edge $x_2 = (t_2, t_3)$ with $d(x_2) < d(x_1)$
  - if none exist, restart with another vertex
  - else we have a solution by removing the edge (t_4, t_3) and connecting (t_1, t_4)

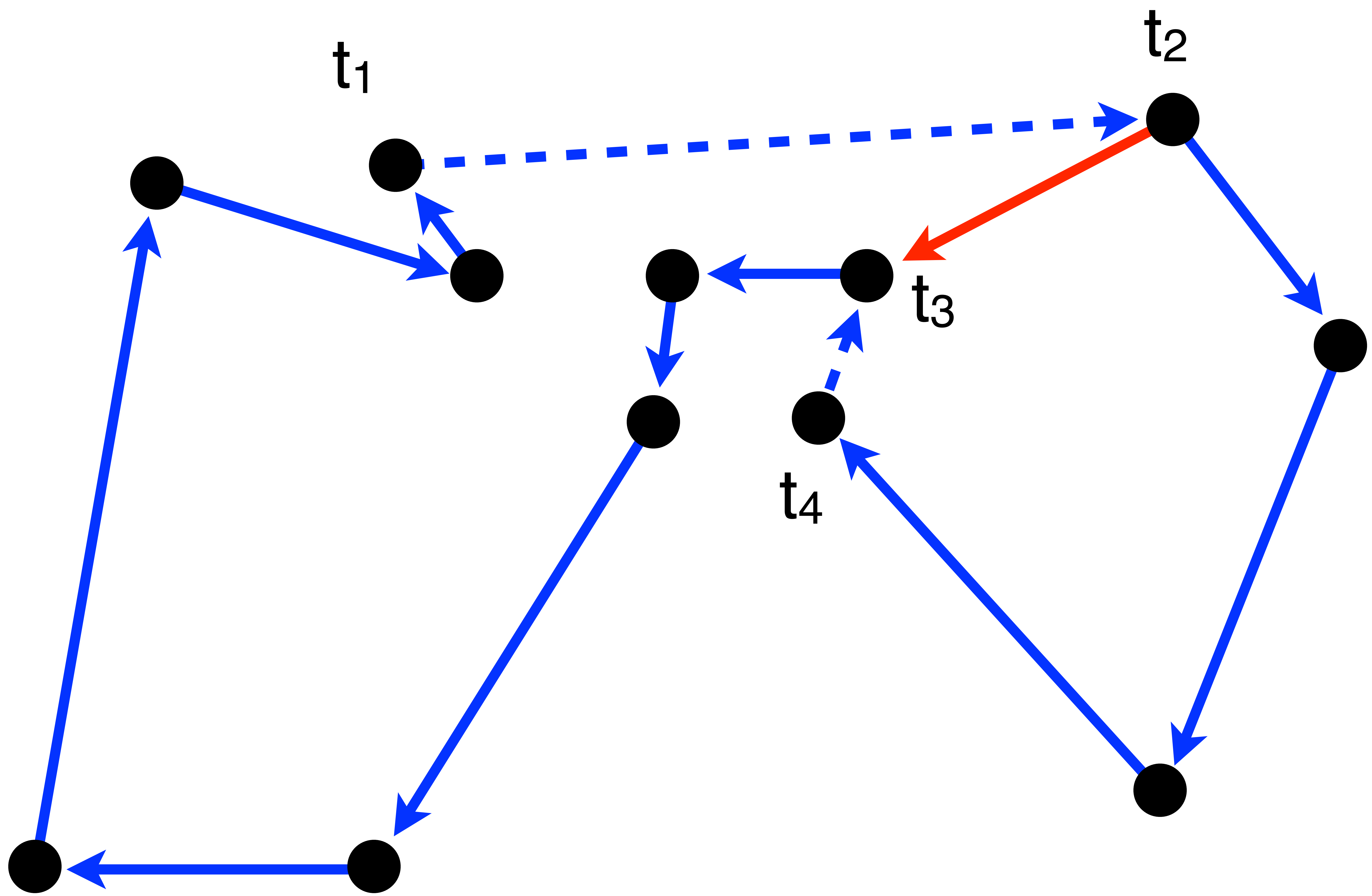
K-OPT (first iteration)



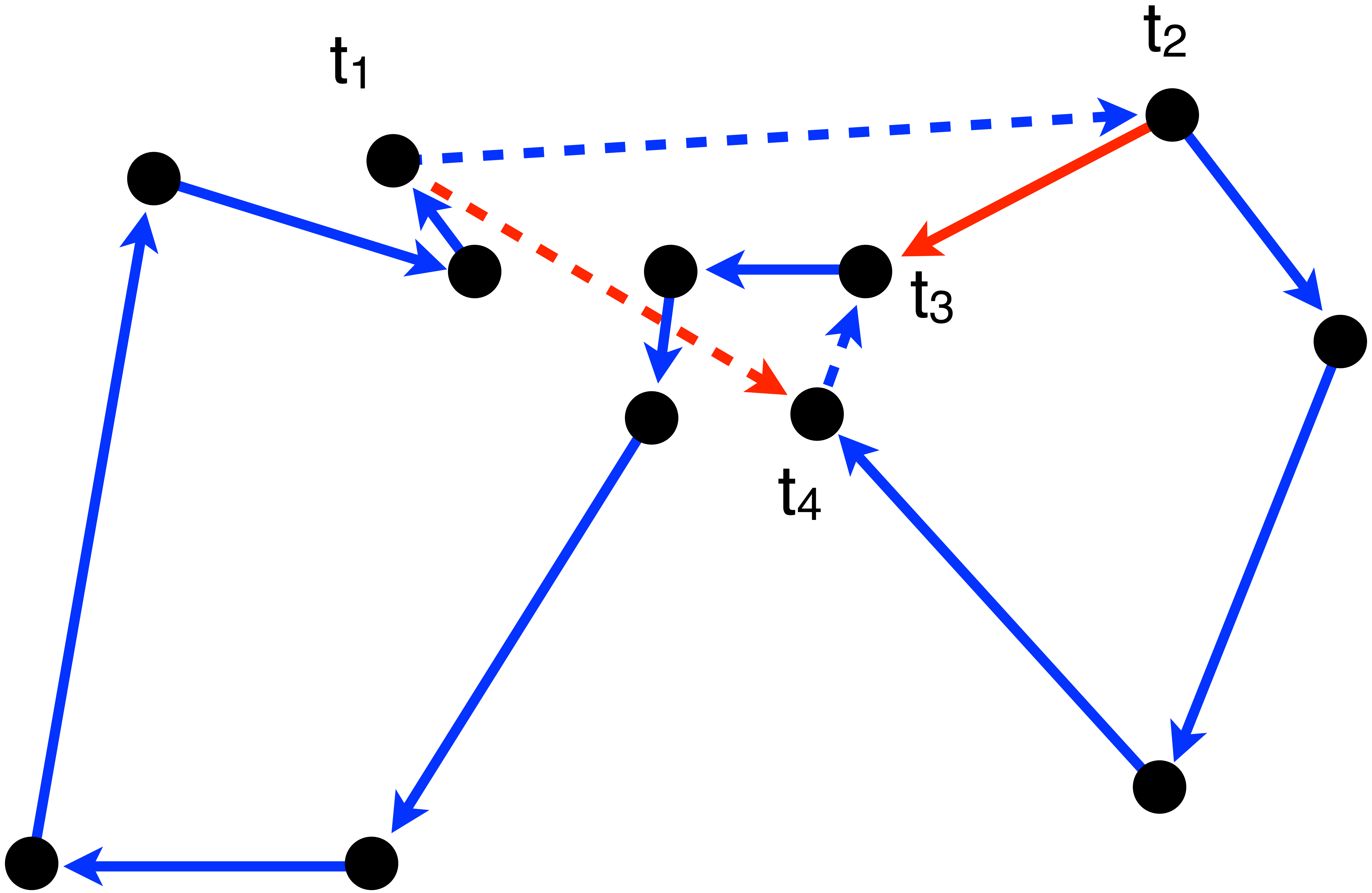
K-OPT (first iteration)



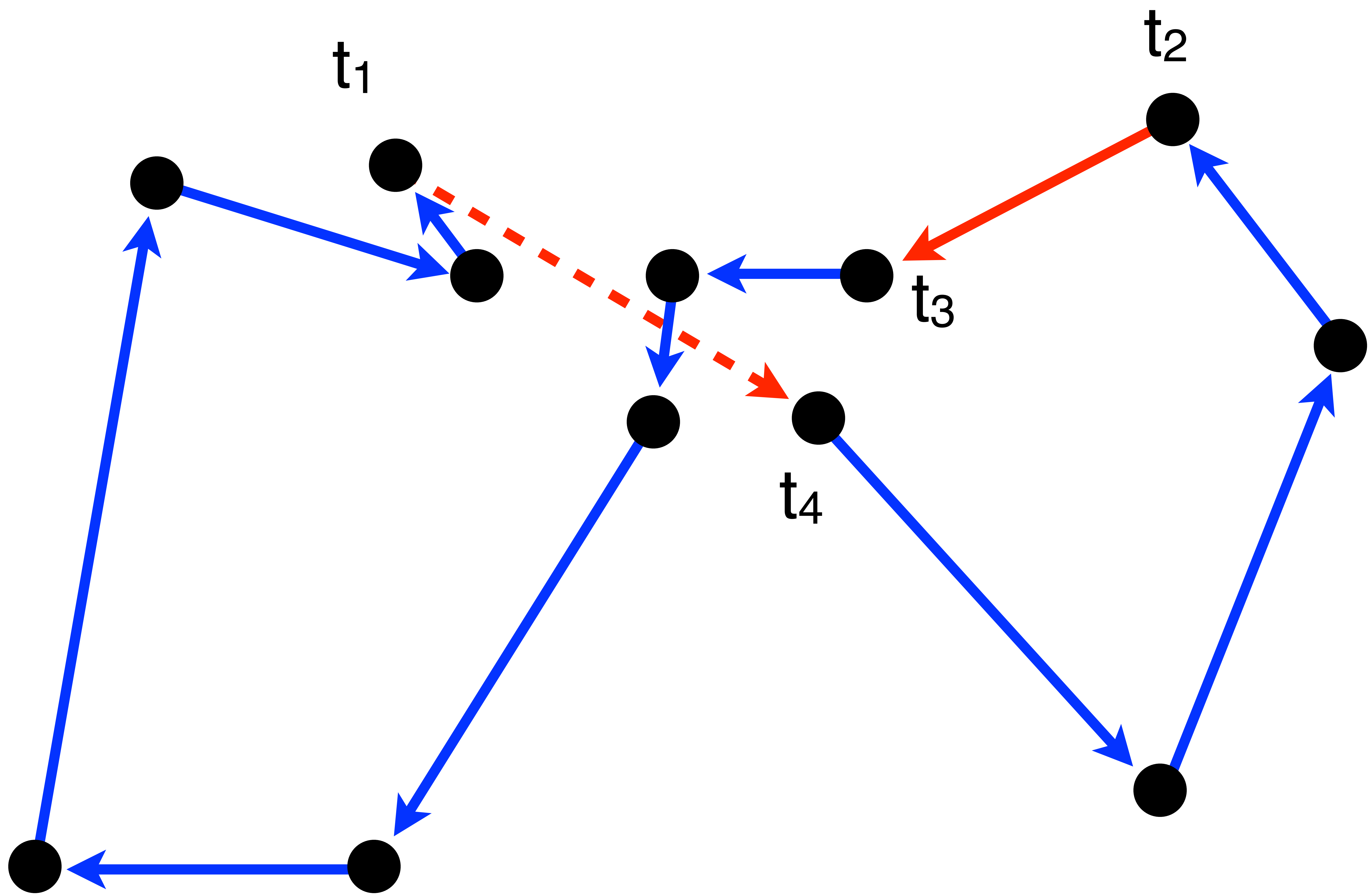
K-OPT (first iteration)



K-OPT (first iteration)



K-OPT (first iteration)



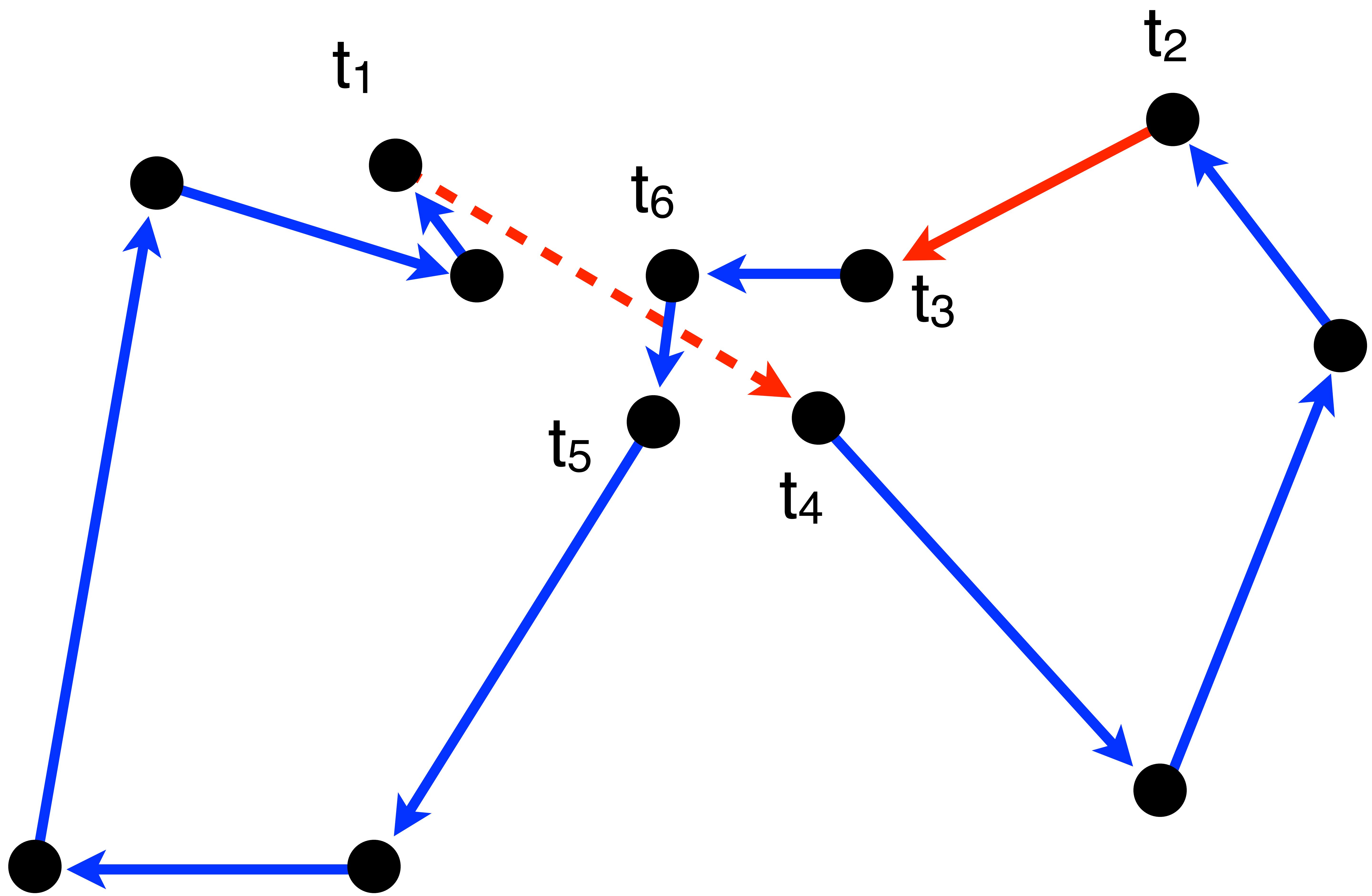
K-OPT

- K-OPT
  - choose a vertex t_1 and its edge $x_1 = (t_1, t_2)$
  - choose an edge $x_2 = (t_2, t_3)$ with $d(x_2) < d(x_1)$
  - if none exist, restart with another vertex
  - else we have a solution by removing the edge (t_4, t_3) and connecting (t_1, t_4)
  - compute the cost but do not connect

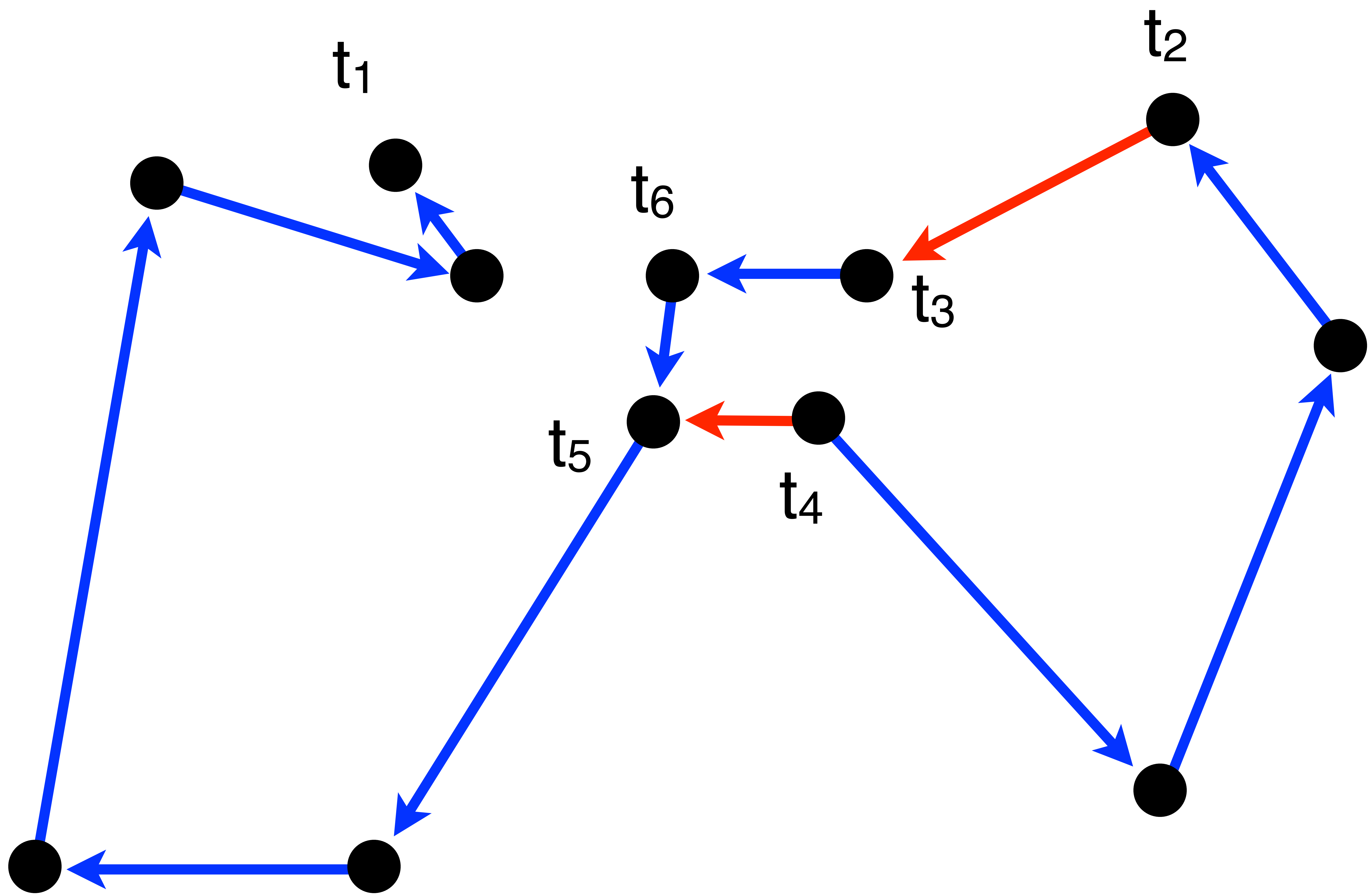
K-OPT

- K-OPT
  - choose a vertex t_1 and its edge $x_1 = (t_1,t_2)$
  - choose an edge $x_2 = (t_2,t_3)$ with $d(x_2) < d(x_1)$
  - if none exist, restart with another vertex
  - else we have a solution by removing the edge (t_4,t_3) and connecting (t_1,t_4)
  - compute the cost but do not connect
  - instead restart with t_1 and its (pretended) edge (t_1,t_4)

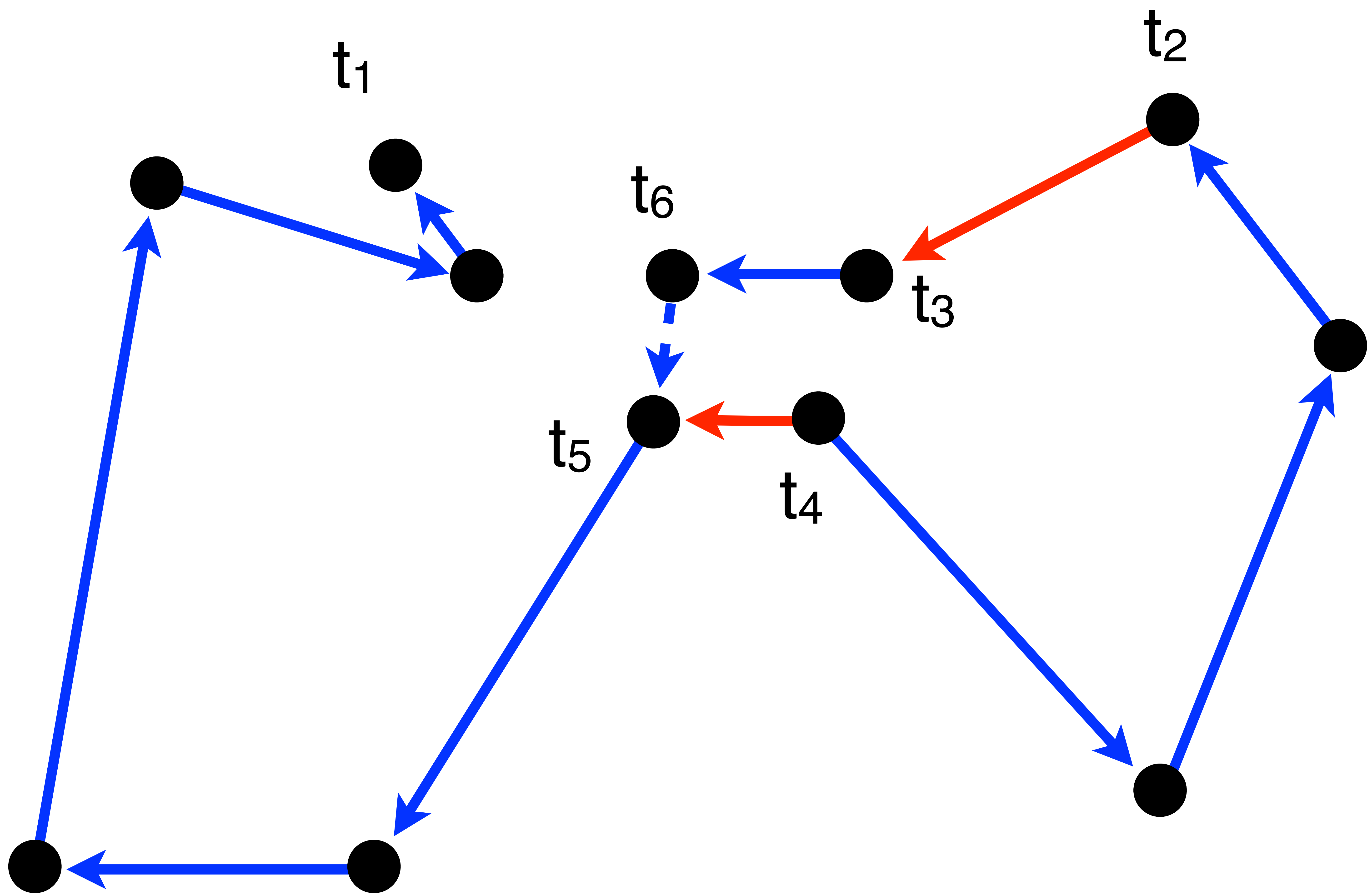
K-OPT (first iteration)



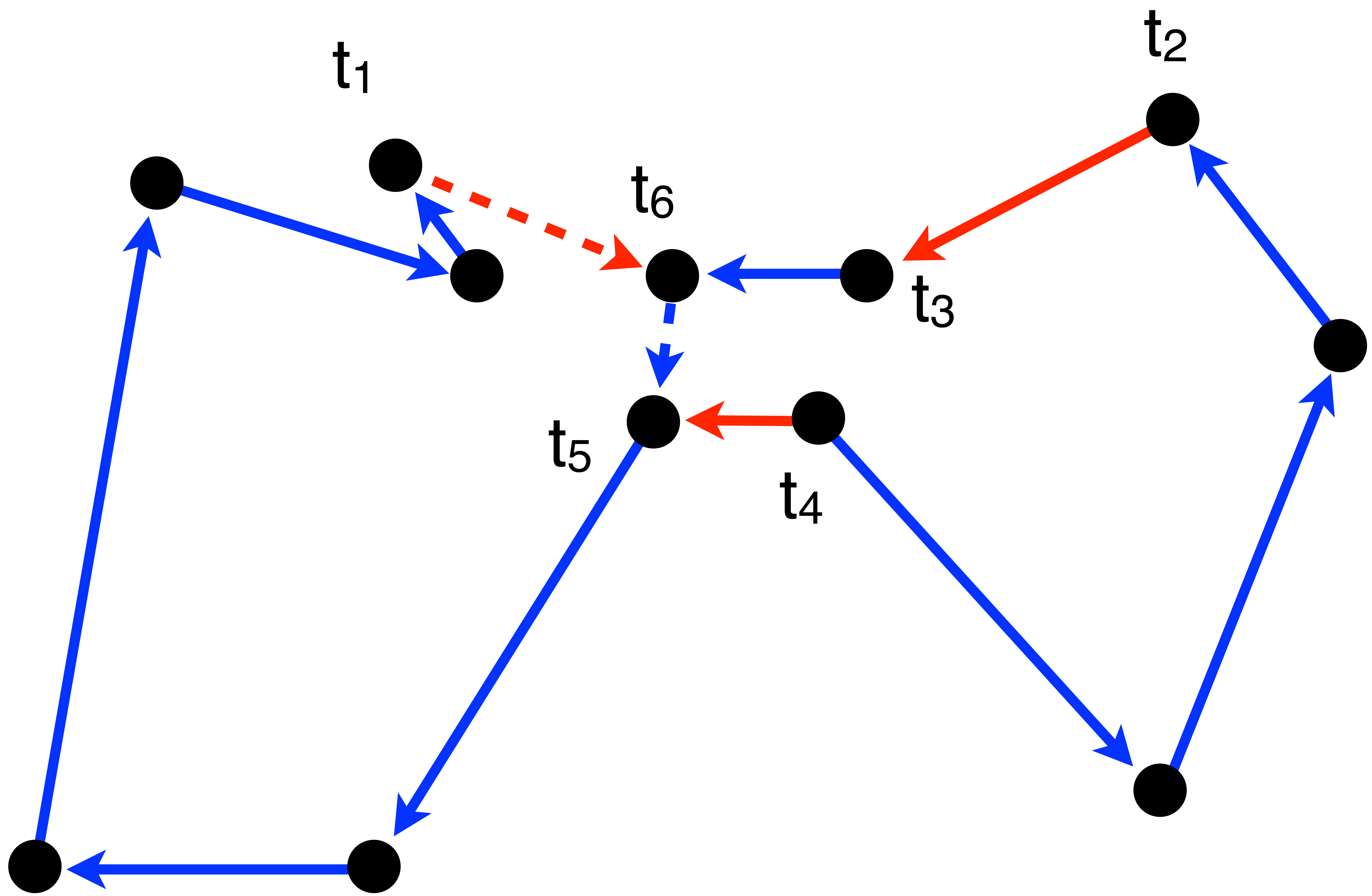
K-OPT (first iteration)



K-OPT (first iteration)



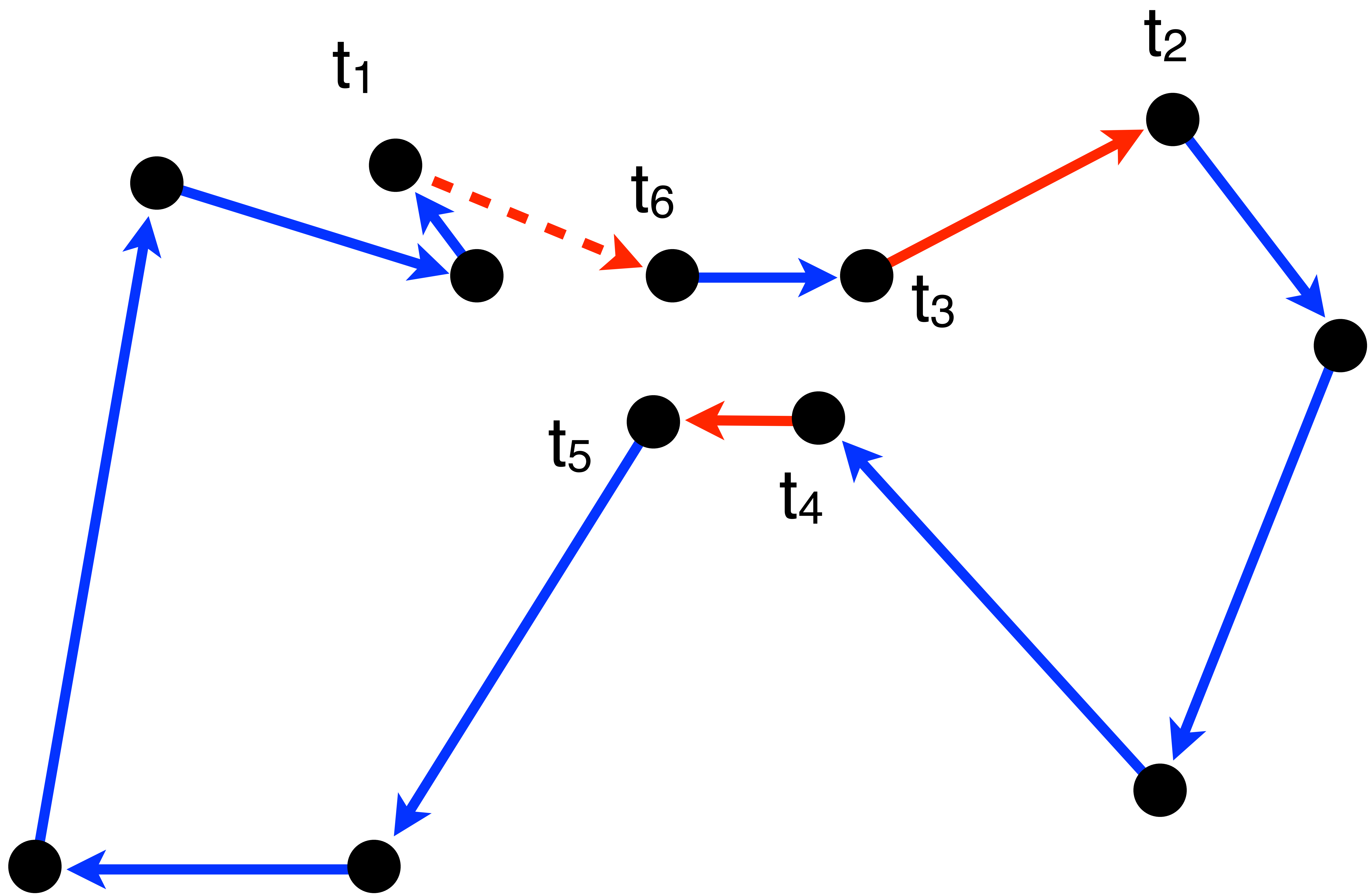
K-OPT (first iteration)



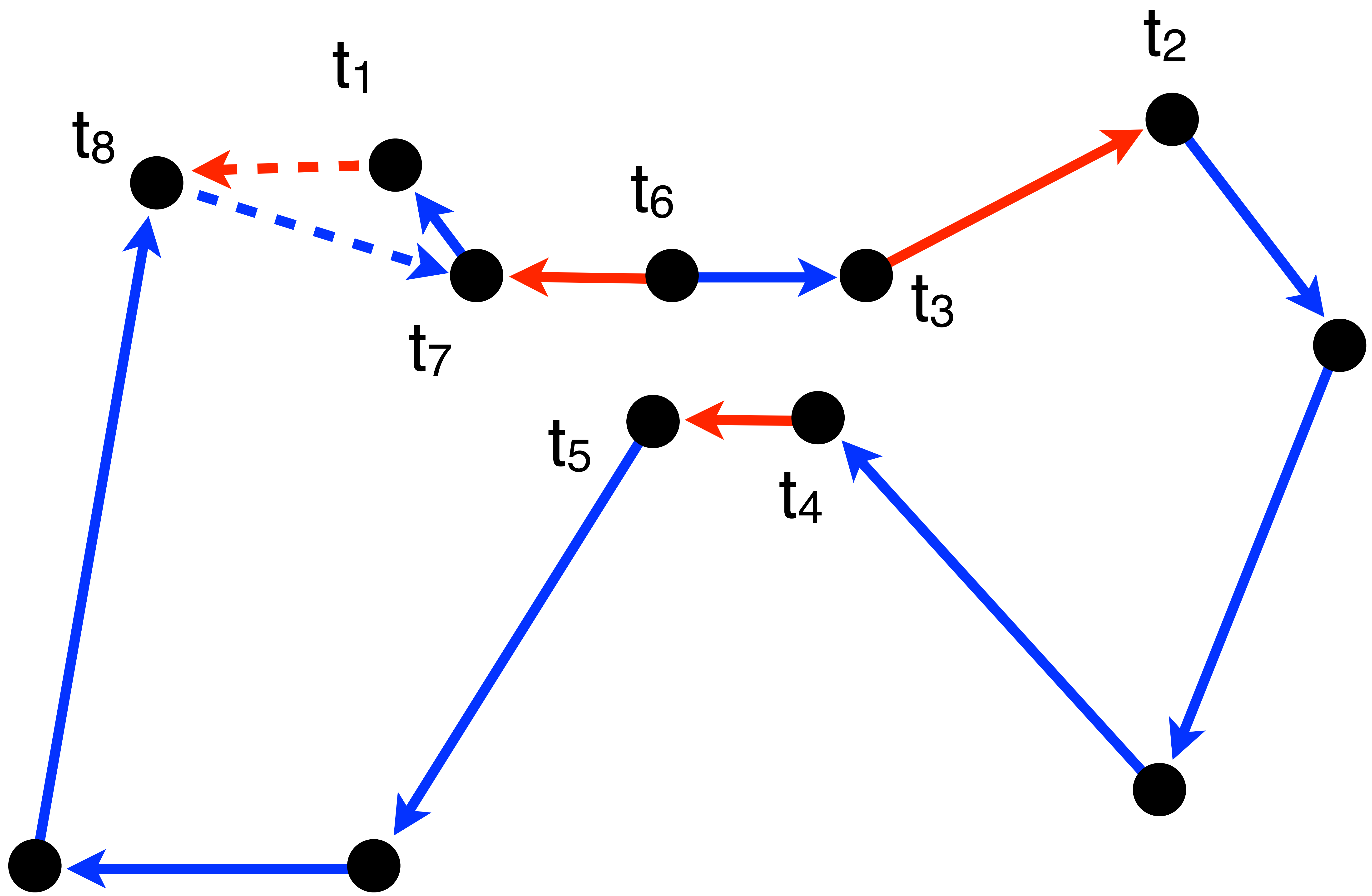
K-OPT

- K-OPT
  - choose a vertex t_1 and its edge $y_1 = (t_1,t_4)$
  - choose an edge $x_2 = (t_4,t_5)$ with $d(y_2) < d(y_1)$
  - if none exist, restart with another vertex
  - else we have a solution by removing the edge (t_6,t_5) and connecting (t_1,t_6)
  - compute the cost but do not connect
  - instead restart with t_1 and its (pretended) edge (t_1,t_6)

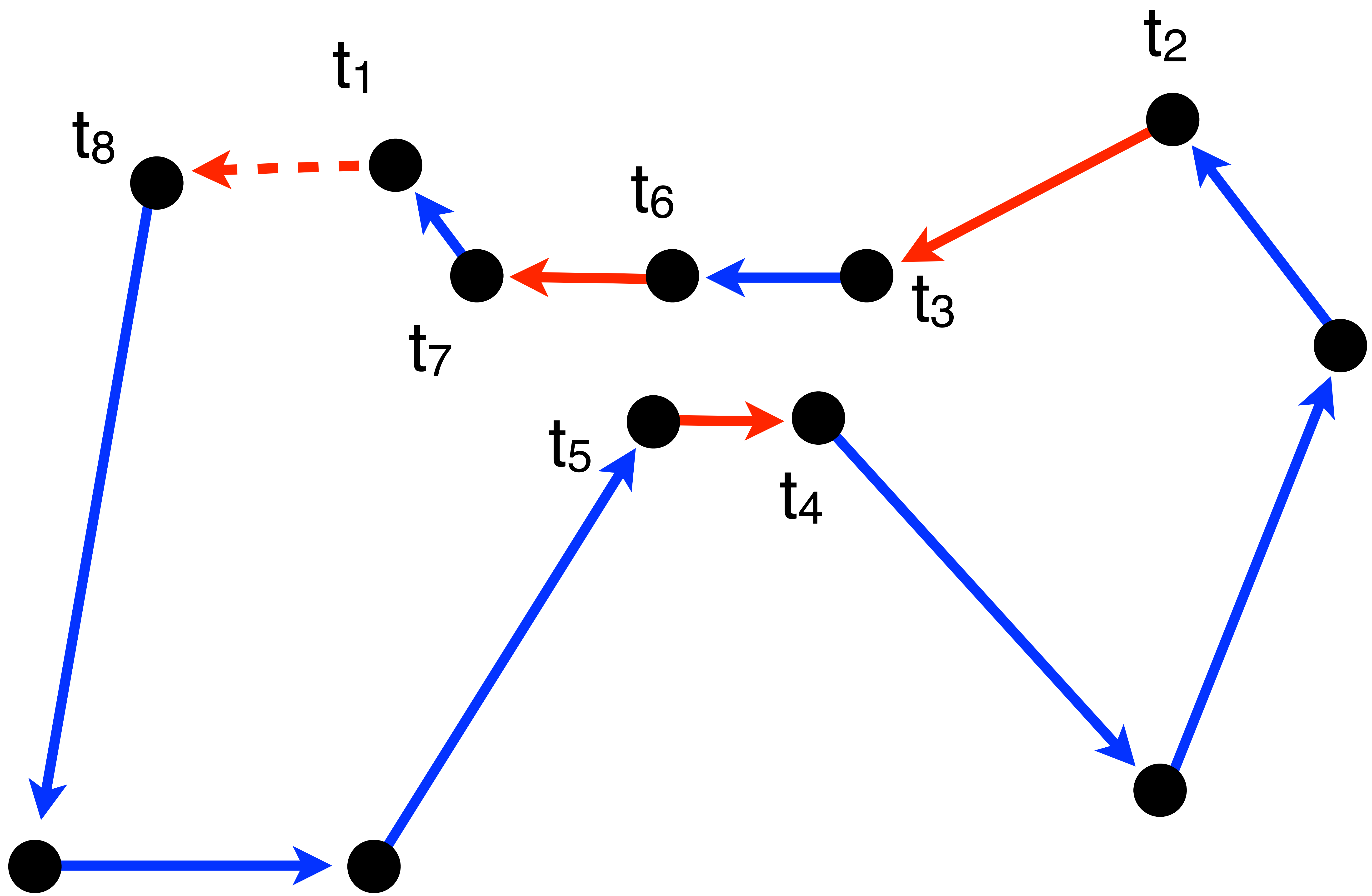
K-OPT (first iteration)



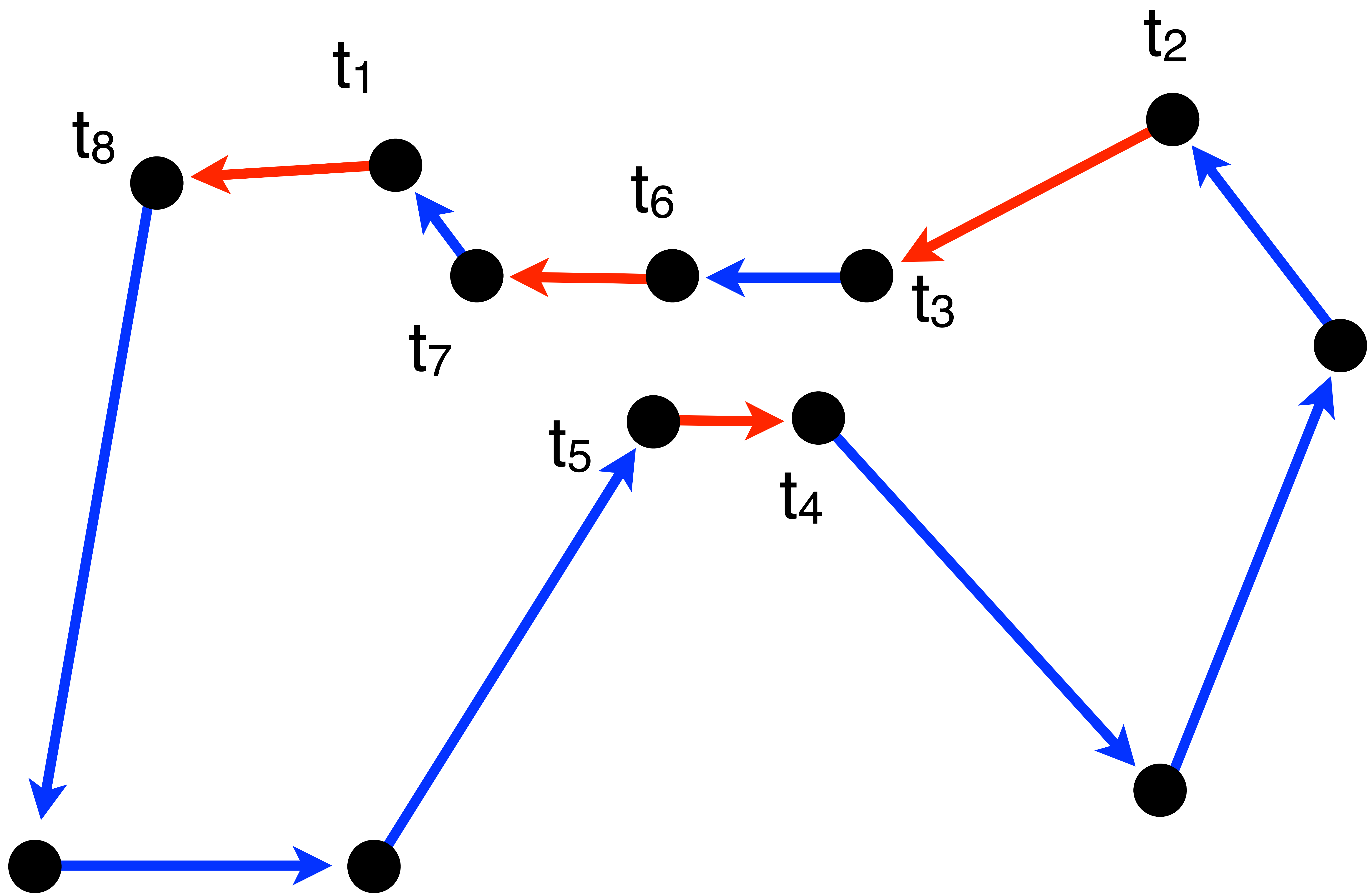
K-OPT (first iteration)



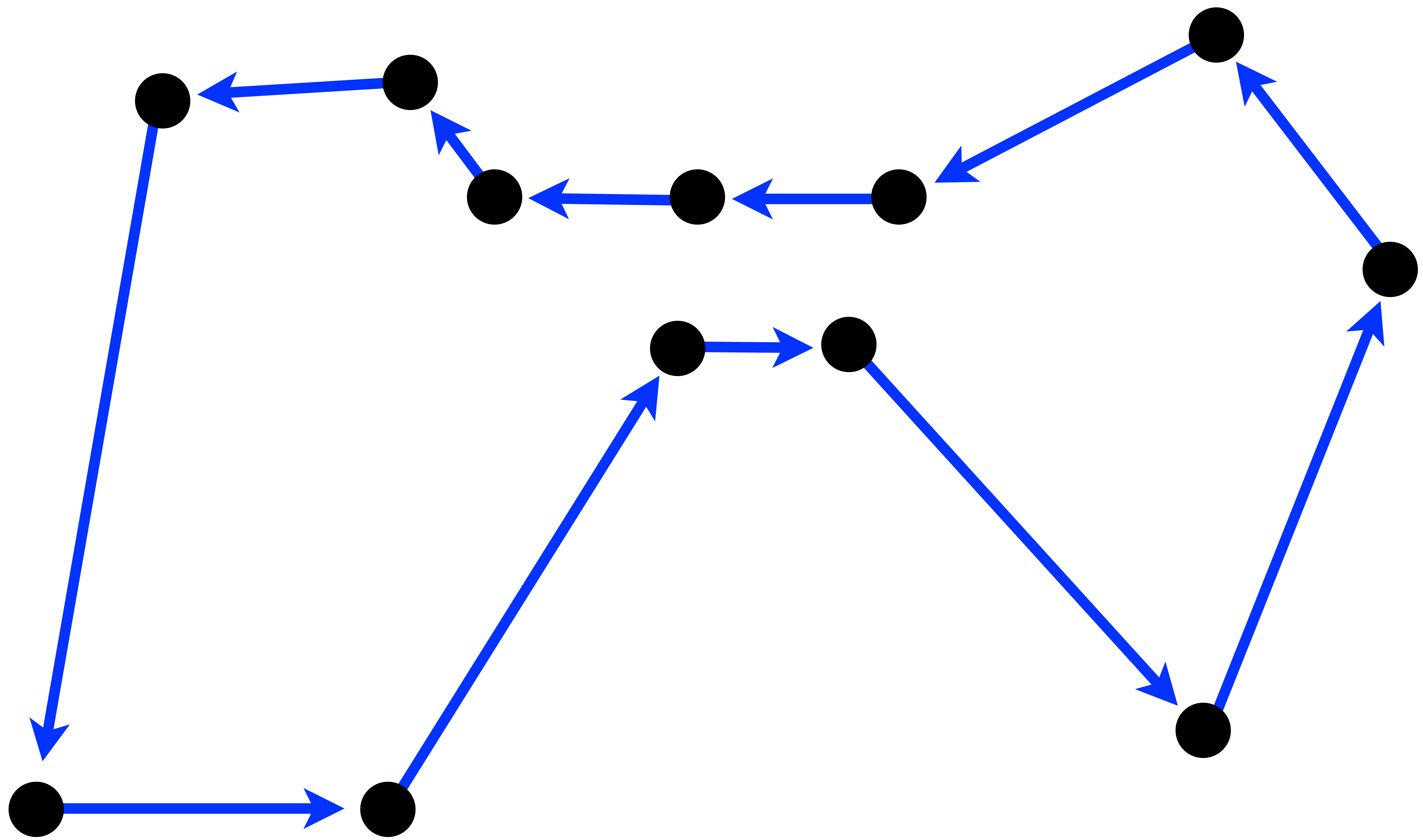
K-OPT (first iteration)



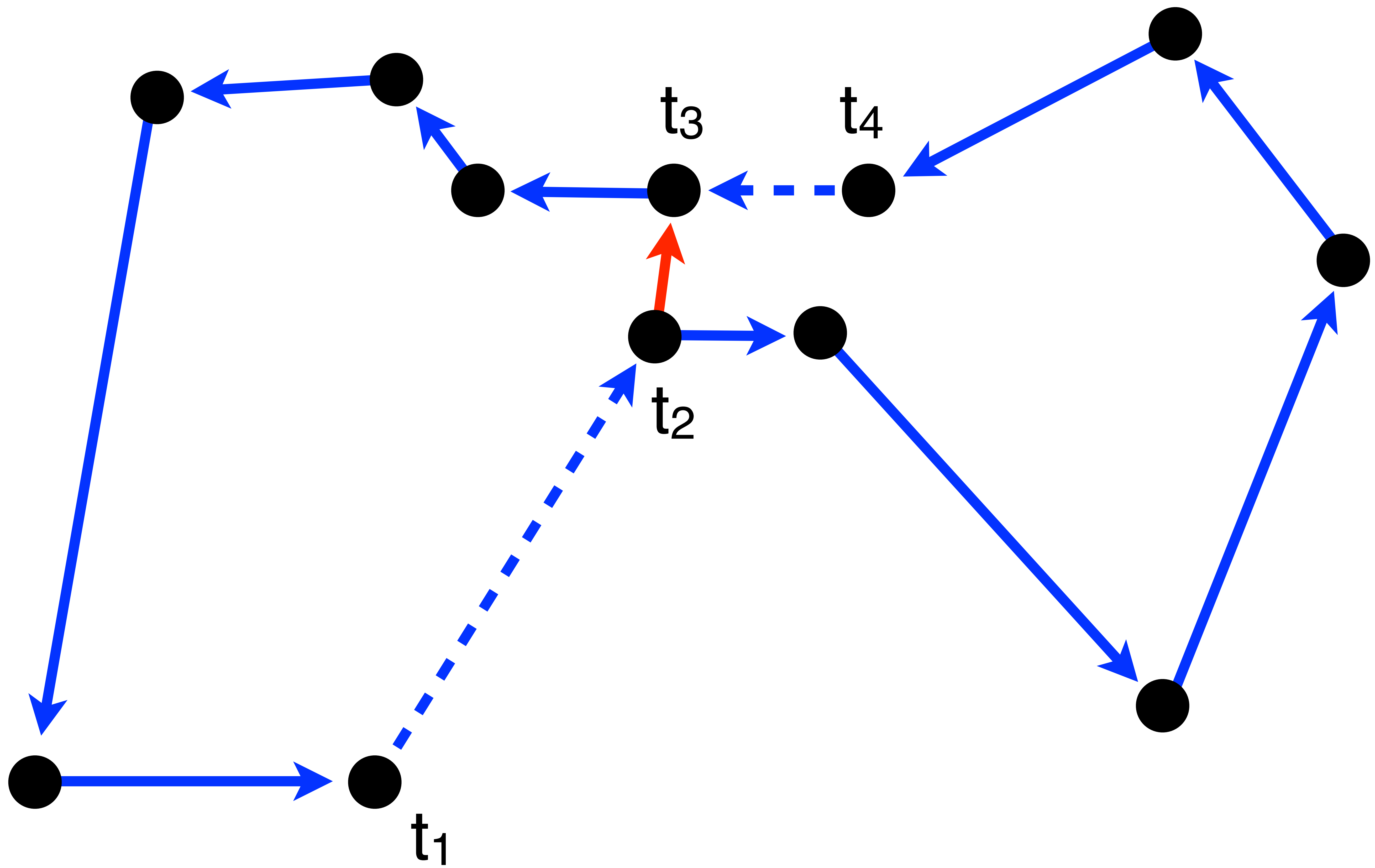
K-OPT (first iteration)



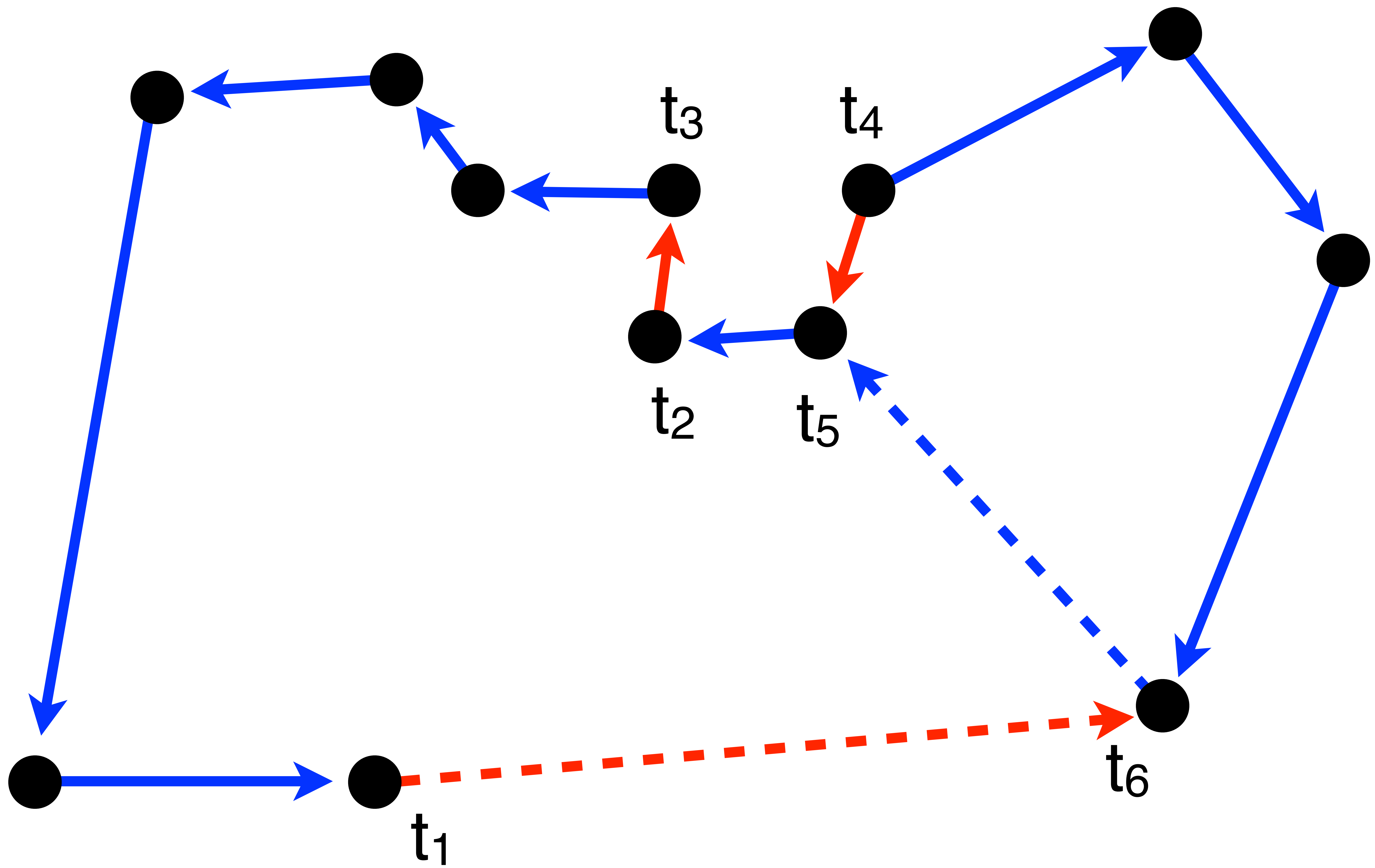
K-OPT (second iteration)



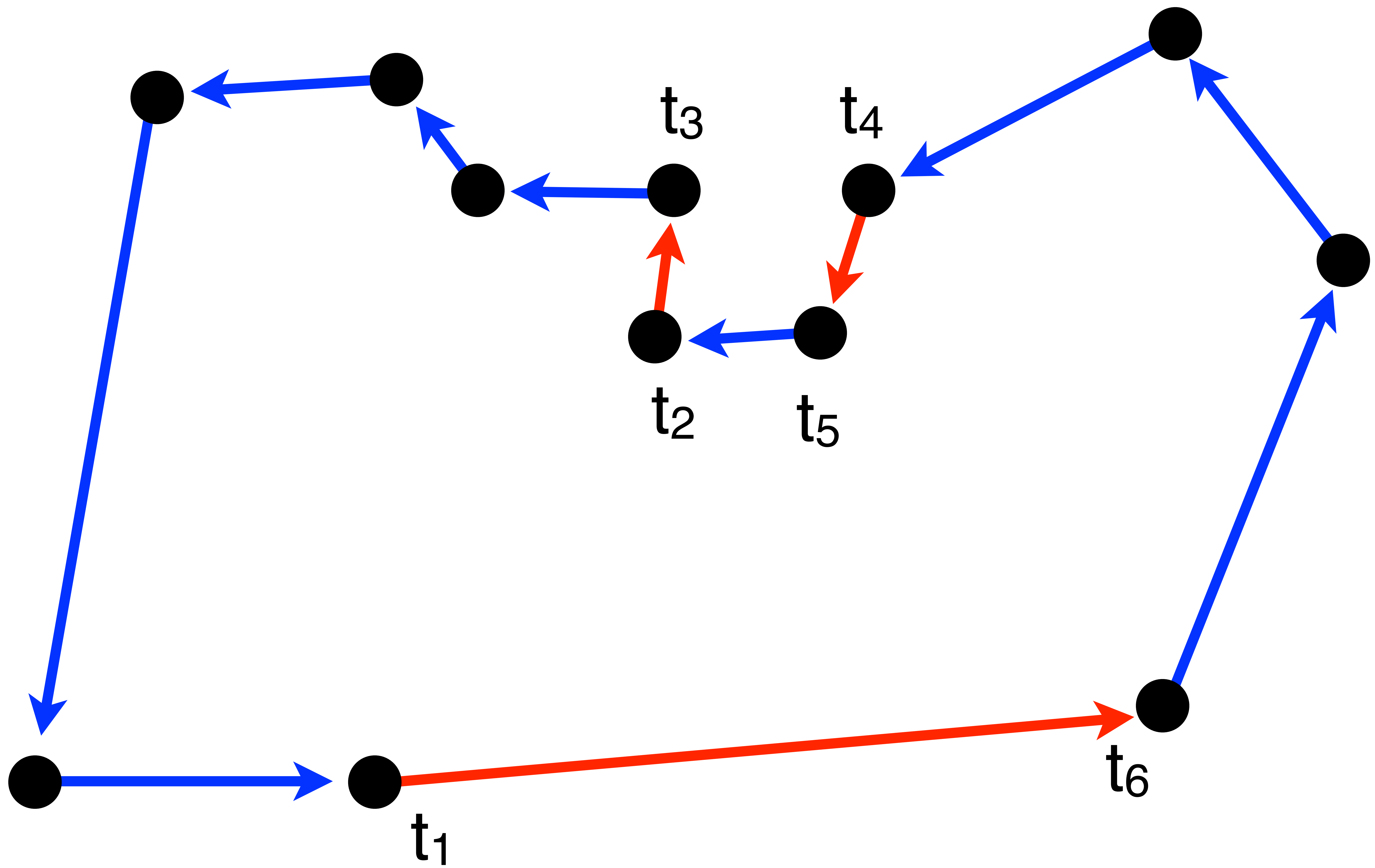
K-OPT (second iteration)



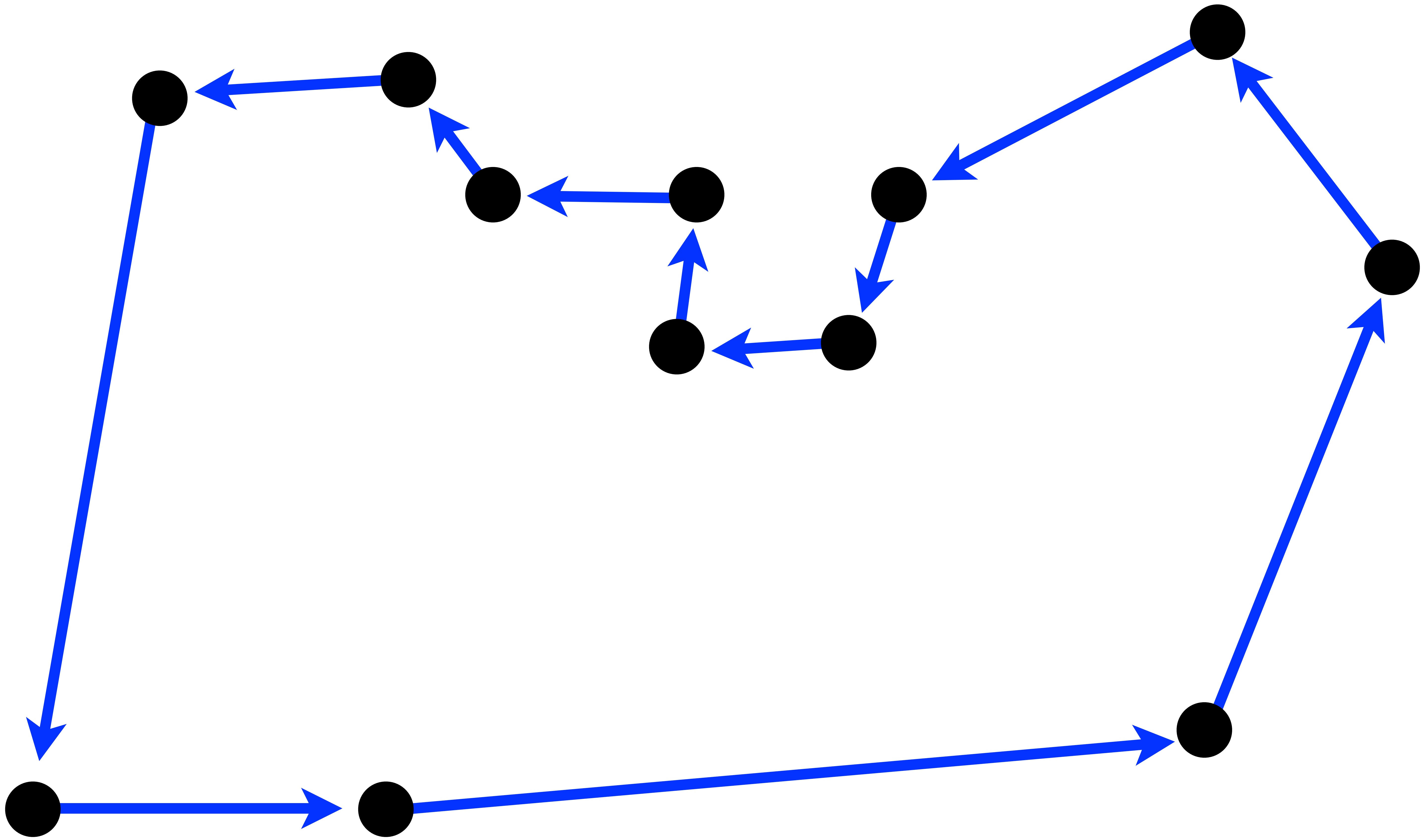
K-OPT (second iteration)



K-OPT (second iteration)



K-OPT (second iteration)



Until Next Time