Discrete Optimization and Integer Programming.

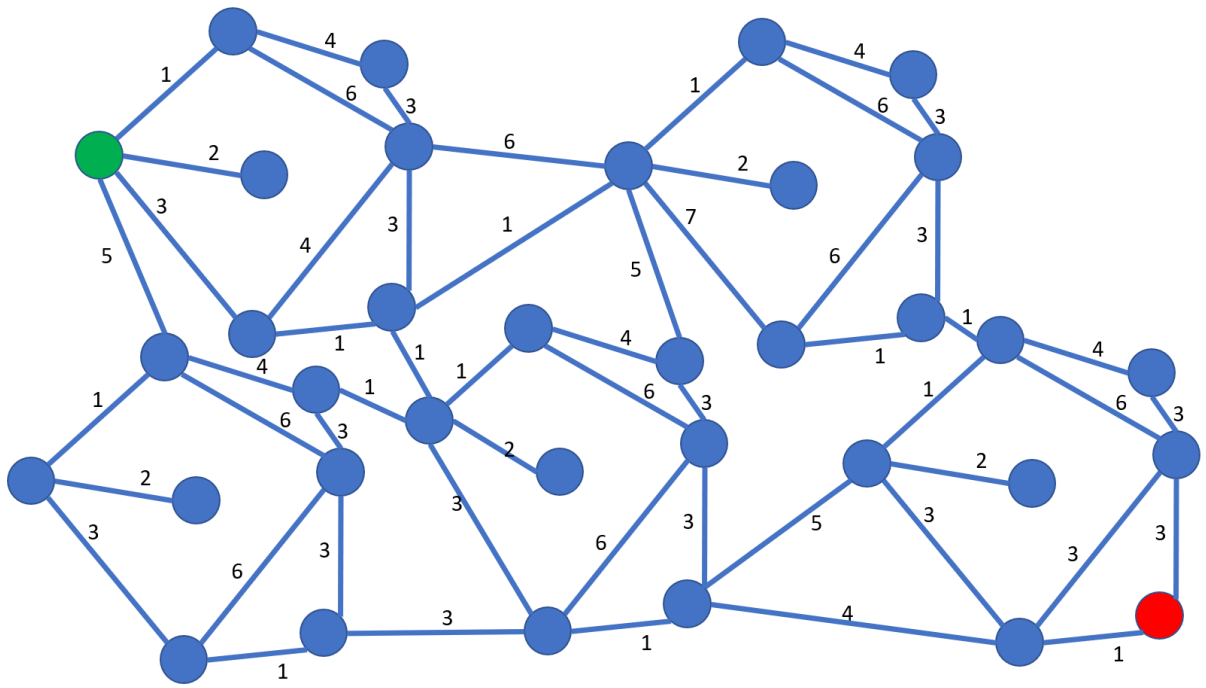
Shortest Path Problem

“Nothing takes place in the world whose meaning is not that of some maximum or minimum.” (L. Euler)

HUAWEI

Contents

I. Shortest Path Problem



We have an directed/undirected graph G with the set of vertices $V(G)$ and the set of edges $E(G)$.

On the set of edges we have the weight function

$$w: E(G) \to \mathbb{R}$$

The weight of a path $P = (e_1, \dots, e_n)$ is defined as the sum of weights of all edges of this path

$$w(P) \coloneqq \sum_{i=1}^{n} w(e_n).$$

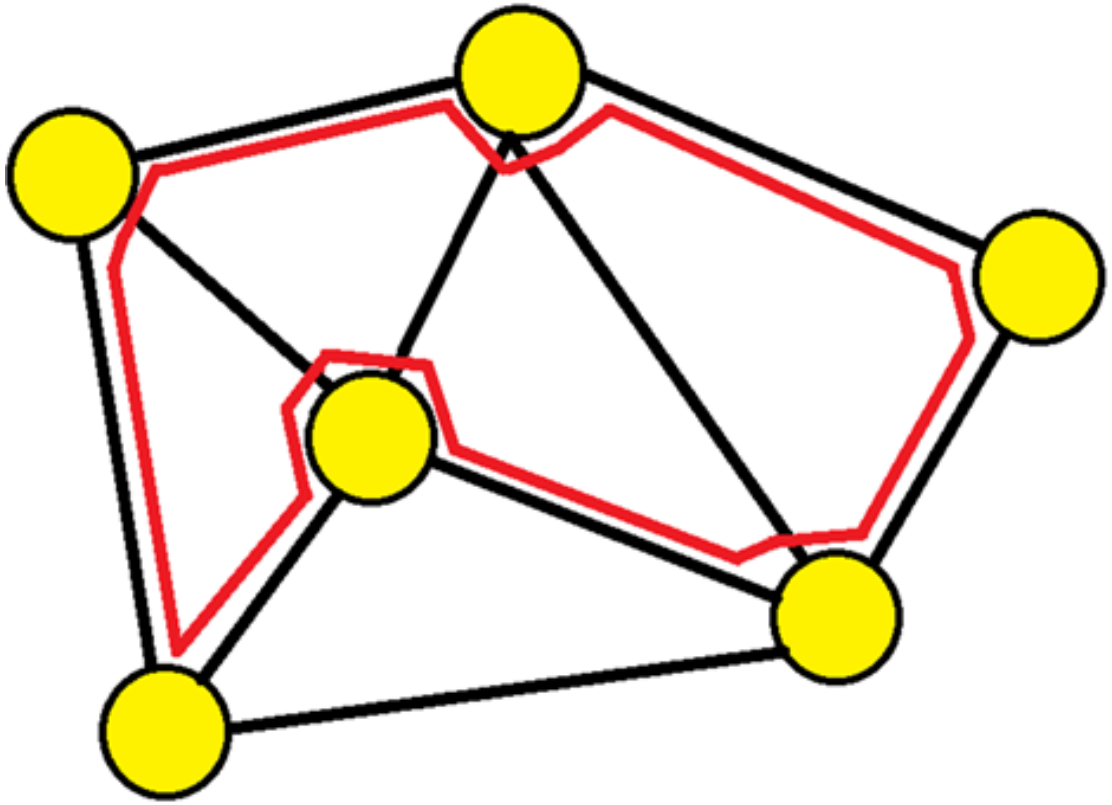
Also we have chosen two nodes which we call the source node and the target node. *The Shortest Path Problem (SPP)* is to find the path between the source and target nodes with minimal weight.

This problem has a huge number of practical applications. In particular, it is often found as a subproblem of more complex combinatorial problems.

I. Shortest Path Problem

The complexity of the problem depends on the weight function w.

For example, suppose that the weights of all links equals to (-1) then the SPP is equivalent to the problem of determination the (topologically) longest path connecting the source and target nodes.
The last problem is NP-hard.

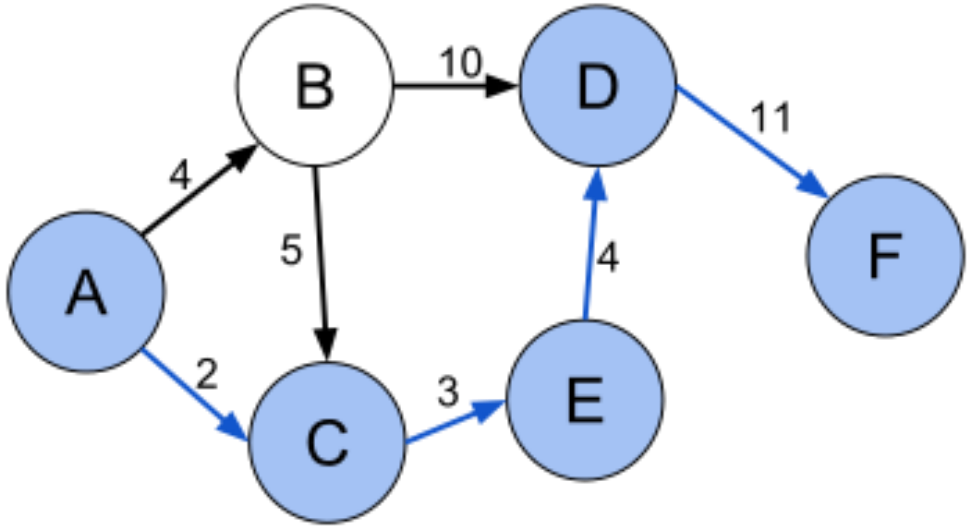


If the weight function does not allow cycles of negative weight in the graph then the SPP can be solved by the *Bellman-Ford algorithm* which is polynomial.

If the weight function of the graph is non-negative then the SPP can be efficiently solved by the *Dijkstra algorithm* which is even faster than the Bellman-Ford algorithm.

I. Dijkstra Algorithm

- ***Description***



Suppose that we have a graph G with non-negative weight function w. Denote a source and target nodes by $s, t \in V(G)$.
In the Dijkstra algorithm we will operate with some functions $l: V(G) \to \mathbb{R}_+ \cup \{\infty\}$, $p: V(G) \to V(G)$ and some set R which is defined during the algorithm.
We want to find out the shortest path between s and t.
The algorithm works as follows

1. Assign $l(s) := 0$ and $l(v) := \infty$ for all $v \in V(G) \setminus \{v\}$. Also $R := \emptyset$.
2. Find a vertex $v \in V(G) \setminus R$ such that $l(v) = \min_{u \in V(G) \setminus R} l(u)$.
3. If $v == t$ then stop the algorithm else go to the following step.
4. Assign $R := R \cup \{v\}$.
5. For all $u \in V(G) \setminus R$: $(v, u) \in E(G)$ do:
   If $l(u) > l(v) + w((v, u))$ then
   assign $l(u) := l(v) + w((v, u))$ and $p(u) := v$.
6. Go to the step 2.

I. Dijkstra's Algorithm

- ***Proof of Correctness***

We need to prove the following statements

1. For all $v \in V(G)\backslash\{s\}$ such that $l(v) < \infty$ we have that

$$p(v) \in R, \qquad w\big(p(v)\big) + w\big((p(v), v)\big) = l(v)$$

and the sequence $v, p(v), p\big(p(v)\big), \ldots$ contains the source node s.

2. For all $v \in R$ we have that

$$l(v) = \mathrm{dist}_{(G,l)}(s, v).$$

Proof of the first statement:

- It is obvious that after the first step of the algorithm these two statements hold.
- If for some $v \in V(G)$ we have that $l(v) < \infty$ then this vertex was updated during the step 5 at some moment.
- Note that in the step 5 the value $l(v)$ reduces to the value $l(u) + w(u, v)$ and $p(v) = u$ only if $u \in R$ and $v \notin R$.
- Since we modify the function p only on the vertices not belonging to R the step 5 respects the property that the sequence $v, p(v), p\big(p(v)\big), \ldots$ contains the source node s.

I. Dijkstra's Algorithm

- ***Proof of Correctness***

Proof of the second statement:

1. The second statement is trivial for the source node s.

2. Suppose that during the step 2 the vertex $v \in V(G)\backslash\{s\}$ is going to be added to R. And also suppose that there is another $s - v$ path P with weight less than $l(v)$.

3. Let y be the first vertex of the path P such that $y \in (V(G) \setminus R) \cup \{v\}$ and let x be the predecessor of the vertex y in the path P.

4. Since $x \in R$ then the step 5 was already applied to this vertex so we have

$$l(y) = l(x) + w((x,y)) = \mathrm{dist}_{(G,l)}(s,x) + w((x,y)) \leq w\left(P_{[s,y]}\right) \leq w(P) < l(v).$$

5. But this contradicts the step 2, so there is no path with weight less than $l(v)$.

I. Dijkstra's Algorithm

• *Complexity*

The complexity of the Dijkstra algorithm depends on the data structures used in its implementation. In particular, it is important how the set R is implemented. The corresponding data structures should support the following three operations

- add an element to R with associated key (weight);
- decrease key of some element from R;
- extract an element from R with minimal key.

Such data structures is called *priority queue*. For implementation of this data structure the running time of the Dijkstra algorithm is the following

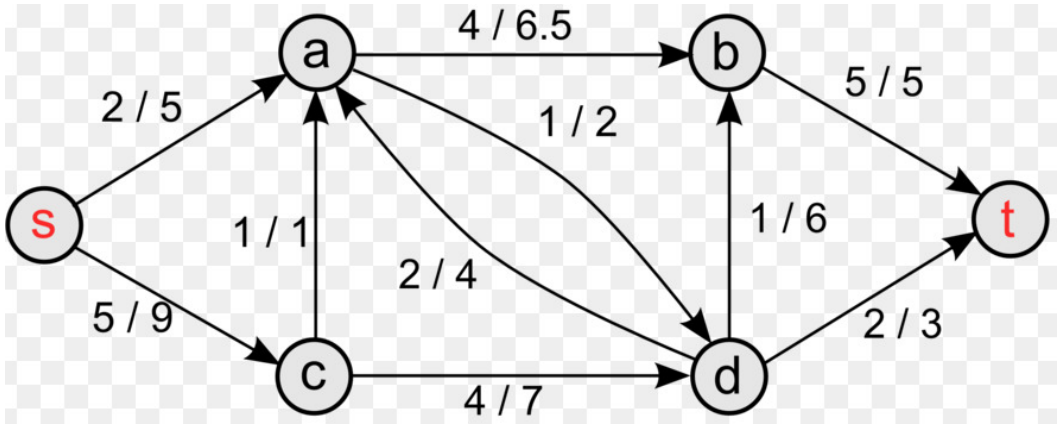
$$O(|E| \cdot T_{\mathrm{dk}} + |V| \cdot T_{\mathrm{em}})$$

where T_{dk} and T_{em} are the complexities of the *decrease-key* and *extract-minimum* operations in *Q*, respectively. Implementation of the priority queue using the so-called *Fibonachi heap* the following running time

$$O(|E| + |V| \log|V|).$$

I. Flow Network

- *A network* is a directed graph G with the set of vertices $V(G)$, the set of edges $E(G)$ and a non-negative function $c: E(G) \to \mathbb{R}_{\geq 0}$ called *the capacity function*. Assume also that we have two nodes s and t called the *source* and *sink*.



- *A flow* with the source node $s \in V(G)$ and the sink node $t \in V(G)$ is a function $f: E(G) \to \mathbb{R}_{\geq 0}$ that satisfies the following constraints
  - For all edges $e \in E(G)$ we have $f(e) \leq c(e)$;
  - For the source node $s \in V(G)$ we have the equality

$$\sum_{e \in E(G) \text{ is outgoing from } s} f(e) > 0;$$

  - For the sink node $t \in V(G)$ we have the equality

$$\sum_{e \in E(G) \text{ is ingoing to } t} f(e) > 0;$$

  - (*flow conservation*/*Kirchhoff's current law*) For all other nodes $v \in V(G) \setminus \{s, t\}$ we have

$$\sum_{e \in E(G) \text{ is ingoing to v}} f(e) = \sum_{e \in E(G) \text{ is outgoing from } s} f(e).$$

I. Min-cost Flow Problem

• *Definitions*

The classical flow network problem is *the max flow problem* which is formulated as follows.

- For given flow network (G, c) we need to find a flow f with the source node s and the sink node t such that *the flow rate* $\sum_{e \in E(G) \text{ is outgoing from } s} f(e)$ is maximal.

This problem can be efficiently solved by *the Ford-Fulkerson algorithm* which is polynomial.

Another classical problem is *the min cost flow problem* which can be formulated as follows.

- Suppose we have a weighted flow network (G, c, w) with the weight function w. The problem is to find a $s - t$ flow f with the fixed flow rate $\sum_{e \in E(G) \text{ is outgoing from } s} f(e) = d$ such that the total cost of the flow $w(f) = \sum_e w(e) f(e)$ is minimal.

The special case of this problem is the shortest path problem considered above.

I. Min-cost Flow Problem

- ***MILP formulation***

The network flow problems can be easily formulated as LP models in the following way.

For simplicity consider a weighted flow network (G, c, w) without multiple links with $V(G) = \{1, \dots, n\}$. Therefore, we have $E(G) \subset V(G) \times V(G)$ so the edges can be parameterized by pairs (i, j).
Let us extended the capacity functions c in the following way

if $(i, j) \notin E(G)$ then $c(i, j) = 0$ and $w(i, j) = 0$.

Also we consider the extended flows f with the property

if $(i, j) \in E(G)$, but $(j, i) \notin E(G)$ then $f(j, i) = -f(i, j)$.

Using this notation the linear program looks as follows

$$f(i, j) = -f(i, j),$$

$$f(i, j) \leq c(i, j),$$

$$\sum_j f(s, j) = d, \qquad \sum_i f(i, t) = d,$$

$$\sum_j f(v, j) = 0 \text{ for all } v \neq s, t,$$

$$\sum_{i,j} w(i, j) f(i, j) \to \min.$$

I. Min-cost Flow Problem

- ***Integral Case***

Now assume that the capacity function of the flow network is equal to 1 on the set of edges of the corresponding graph. One can see that the min-cost flow with the flow rate equal to 1 traverses through a shortest path between s and t.
So in this case the Min-Cost Flow Problem is equivalent to the Shortest Path Problem.

Therefore, we can solve the Shortest Path Problem using MILP.

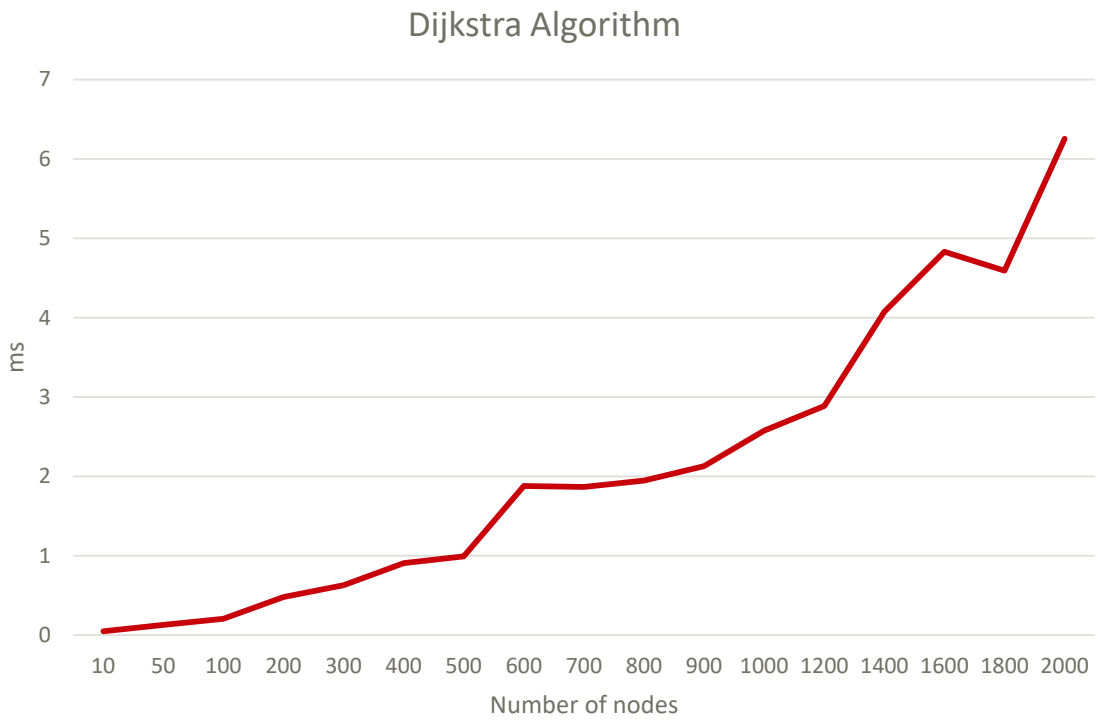
Moreover, one can show that if the flow problem is integral then the corresponding matrix of LP model is totally unimodular which means that LP solution of this model is integral in fact. So we can solve the Shortest Path Problem using any LP algorithm (e.g., simplex method).

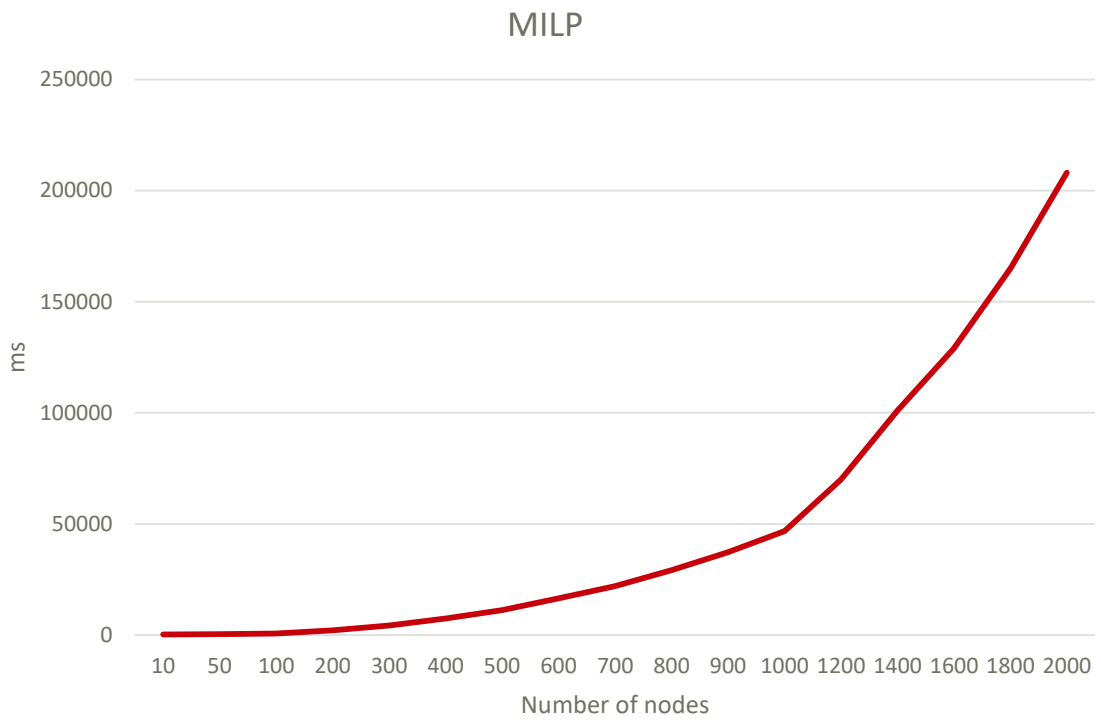
Similarly, if we consider the capacity function of the flow network is equal to n and the flow rate 1 then the corresponding min-cost flow gives n disjoint paths which connect the same pair of vertices and have minimum total weight. In particular, it shows that the problem of finding such paths can be solved for polynomial time.

I. Min-cost Flow Problem

- ***Comparison with Dijkstra***

The graphs below depict time performance of the Dijkstra algorithm and the CBC solver. We see that the Dijkstra algorithm works much faster than the MILP solver.
The presented time performance of the solver is unacceptable in practice.





Thank you.

Bring digital to every person, home, and organization for a fully connected, intelligent world.