Combinatorial Optimization

Zdeněk Hanzálek
hanzalek@fel.cvut.cz

CTU FEE Department of Control Engineering

May 3, 2013

European Social Fund Prague & EU: We invests in your future.

Shortest Paths

Zdeněk Hanzálek
hanzalek@fel.cvut.cz

CTU FEE Department of Control Engineering

April 15, 2013

Table of contents

Problem Statement

a) Shortest Path

- **Instance:** Digraph G, weights $c : E(G) \to \mathbb{R}$, nodes $s, t \in V(G)$.
- **Goal:** Find shortest $s - t$ path P, i.e. one of minimum weight $c(E(P))$, or decide that t is unreachable from s.

Another problems involving the shortest path:

b) from source node s to every node (**Shortest Path Tree - SPT**)

c) from every node to sink node t

d) between all pairs of nodes (**All Pairs Shortest Path**)

Problem a) is often solved by algorithms for b), c) or d). There is no known algorithm with a better asymptotic time complexity. The algorithm can be terminated when t is reached.

Problem c) can be easily transformed to b) by reversing the edges.

Similar Problems

1) The **longest path** can be transformed to the shortest path while reversing the sign of weights. Thus, we search for the minimum instead of the maximum.

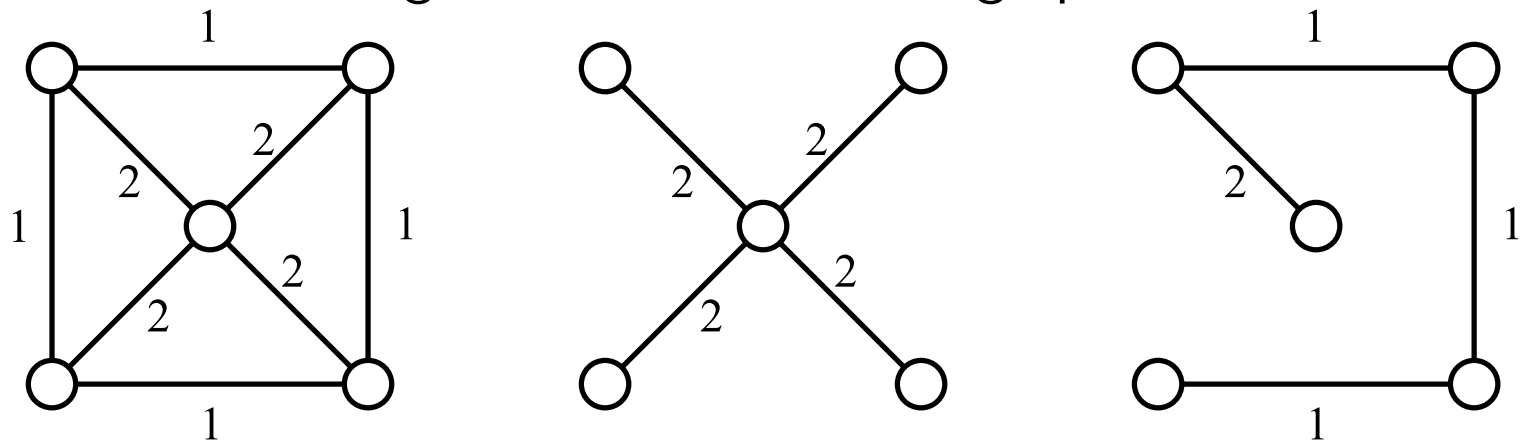
2) When the **nodes are weighted**, or both nodes and edges are weighted, the weight of the path is the sum of the edges and the nodes weights along this path. This can be transformed to the weighted edges case as follows:

- **Replace** every node v from the original graph by the **pair of nodes** v_1 and v_2 and connect them by an edge with weight equal to the weight of node v.
- the edge entering node v now enters v_1
- the edge leaving node v now leaves v_2

Different Problems

1) **Minimum Spanning Tree - MST**

In an undirected graph with the weights associated to arcs, find a spanning tree of minimum weight or decide that the graph is not connected.



The spanning tree in the middle (SPT from central node) has weight 8 and radius 4 (the longest path between two nodes) while the spanning tree on the right side (MST) has weight 5 and radius 5.

2) **Steiner Tree**

Given a connected undirected graph G, weights $c : E(G) \to \mathbb{R}^+$, and a set $T \subseteq V(G)$ of terminals, find a Steiner tree for T , i.e. a tree S with $T \subseteq V(S)$ and $E(S) \subseteq E(G)$, such that $c(E(S))$ is minimum.

Negative Weights YES and Negative Circuits NO

We consider an oriented graphs:

- negative weights are allowed
- negative circuits are not allowed, since the shortest path problem becomes **NP-hard** when the graph contains a negative circuit,

When we transform **undirected graph** to directed graph, then we consider only instances with **nonnegative weights**:

- every undirected edge connecting v and w is transformed to two edges (v,w) and (w,v)
- this transformation of the negative undirected edge results in a negative circuit

Edge Progression and Path

Theorem - existence of the shortest path

If a path from s to t exists in the graph, then the shortest path from s to t exists too.

Note: the **theorem does not hold for edge progression** - in a graph with a negative circuit we can always find an even shorter edge progression which repeatedly goes through this negative circuit.

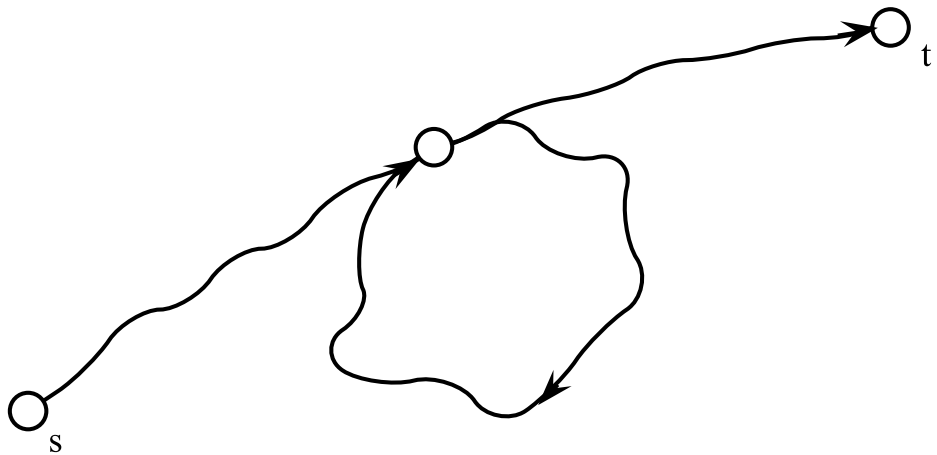
Definitions

- the **Length of the path** P is the **sum of the weights** of its edges (will be denoted simply as $c(E(P))$).
- $l(s, t)$, a distance from s to t, is defined as a length of the **shortest** path from s to t.

Edge Progression and Path

Theorem - shortest edge progression and path

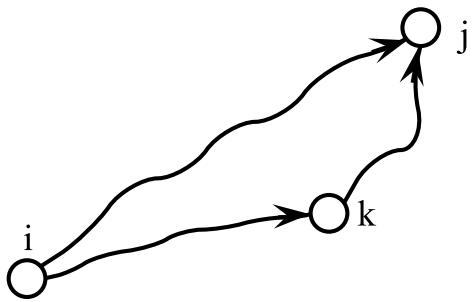
- If there is no **negative weight or zero weight circuit** in the graph, then every shortest edge progression from s to t is the shortest path from s to t.
- If there is no **negative weight circuit** in the graph, then every shortest edge progression from s to t contains the shortest path from s to t and the length of this path is the same.



Triangle Inequality of Shortest Paths

Theorem - triangle inequality

If the graph does not contain a circuit of negative weight then distances between **all triplets of nodes** i, j, k satisfy: $l(i,j) \leq l(i,k) + l(k,j)$.



Corollary: Let $c(i,j)$ **be the weight of edge** from i to j.
Then if the graph does not contain a negative circuit:
$l(i,j) \leq c(i,j)$, $l(i,j) \leq l(i,k) + c(k,j)$ and $l(i,j) \leq c(i,k) + l(k,j)$

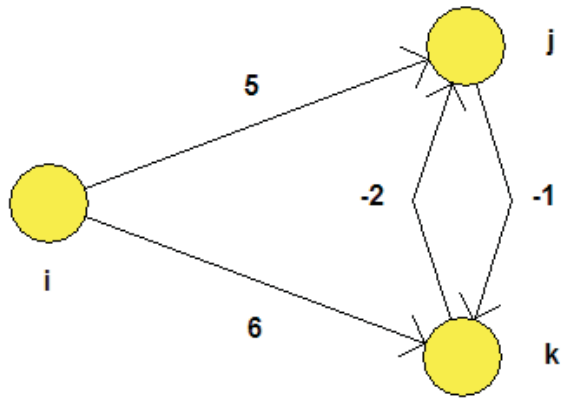
Basic Facts - Negative Circuit

The algorithms listed below make use of the following

- (1) Their computational speed is based on the fact that they **do not care about the repetition of nodes along the path** (i.e. they do not distinguish the path from edge progression).
- (2) If the graph contains a negative circuit, **(1) can not be used** because the shortest edge progression does not need to exist (then it is NP-hard to find the shortest path).

Example: a graph with a negative circuit - consequently, the triangular inequality does not hold - the negative circuit of the edge progression is created while joining the two shortest paths.

$$
\begin{array}{rcccl}
l(i,j) & \leq & l(i,k) & + & l(k,j) \\
6-2 & \leq & (5-1) & + & (-2) \\
4 & \leq & 2 & \ldots & \text{contradiction}
\end{array}
$$



What the graph can not contain, when interested in the longest path?

Bellman's Principle of Optimality

Theorem - Bellman's Principle of Optimality

Suppose we have a graph without negative circuits. Let $k \in \mathbb{N}$, and let s and w be two vertices. Let P be a shortest one among all s-w-paths with at most k edges, and let $e = (v, w)$ be its final edge. Then $P[s, v]$ (i.e. P without the edge e) is a shortest one among all s-v-paths with at most $k - 1$ edges.

Proof by contradiction: Suppose Q is a shorter s-v-path than $P[s, v]$, i.e. $c(E(Q)) < c(E(P[s, v]))$, and $|E(Q)| \leq k - 1$.

- If Q does not contain w, then s-w-path consisting of $Q + e$ has length $c(E(Q)) + c(e) < c(E(P[s, v])) + c(e) = c(E(P))$.
- If Q does contain w, then $Q[s, w]$ has length $c(E(Q[s, w])) = c(E(Q)) + c(e) - c(E(Q[w, v] + e)) < c(E(P))$, because $Q[w, v] + e$ is a non-negative circuit.

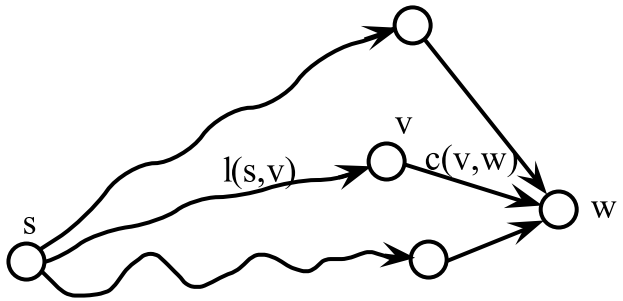
In both cases we have a contradiction to the assumption that P is a shortest s-w-path with at most k edges.

Generalization - Shortest Path Consists of Shortest Paths

Proposition - the shortest path consists of the shortest paths

Suppose we have a graph without negative circuits. If the shortest path from s to w contains node v, the segment of the path from s to v is the shortest s-v-path and similarly the segment of the path from v to w is the shortest v-w-path and $l(s,w) = l(s,v) + l(v,w)$.

Corollary: (**Bellman's equation**) $l(s,w) = \min_{v \neq w} \{l(s,v) + c(v,w)\}$ holds if the graph does not contain a negative circuit



Dijkstra's Algorithm [1959] - Nonnegative Weights

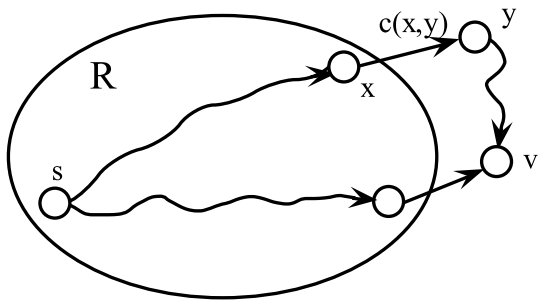
Input: digraph G, weights $c : E(G) \to \mathbb{R}_0^+$ and node $s \in V(G)$.
Output: Vectors l and p. For $v \in V(G)$, $l(v)$ is the length of the shortest path from s and $p(v)$ is the previous node in the path. If v is unreachable from s, $l(v) = \infty$ and $p(v)$ is undefined.

$l(s) := 0$; $l(v) := \infty$ for $v \neq s$; $R := \emptyset$;
while $R \neq V(G)$ **do**
    Find $v \in V(G) \setminus R$ such that $l(v) = \min_{w \in V(G) \setminus R} l(w)$;
    $R := R \cup \{v\}$;
    // calculate $l(w)$ for all nodes on border of R
    for $w \in V(G) \setminus R$ such that $(v, w) \in E(G)$ **do**
        if $l(w) > l(v) + c(v, w)$ **then**
            $l(w) := l(v) + c(v, w)$; $p(w) := v$;
        end
    end
end

Correctness of Dijkstra's Algorithm

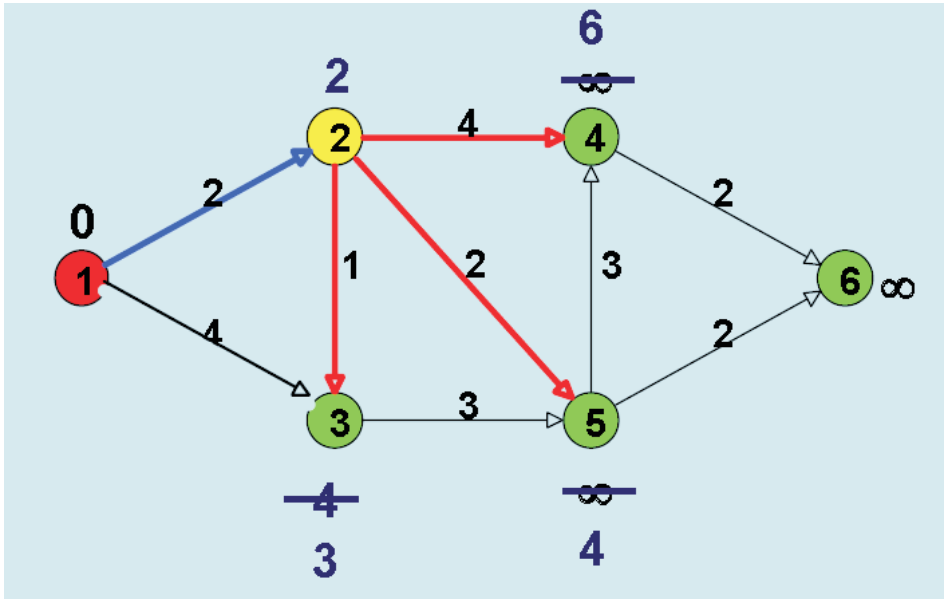
Proof by induction: Inductive assumption - all nodes in R have optimal value of the shortest path. For $|R| = 1$ it is trivial.
We prove inductive step, i.e. the $l(v)$ value is **permanent** (optimal) when we add the **best candidate** v to set R.
The value of $l(v)$ can not be augmented by the path going through $y \in V(G) \setminus \{R \cup v\}$ since:

- $l(x) + c(x, y) \geq l(v)$ due to internal loop of Dijkstra's algorithm
- the weights of all edges from y to v are nonnegative.



Dijkstra is so-called **label setting algorithm**, since label $l(v)$ becomes permanent (optimal) at each iteration. In contrast, **label-correcting algorithms** (e.g. Bellman-Ford's or Floyd's algorithm) consider all labels as temporary until the final step, when they all become permanent.

Iteration of Dijkstra's Algorithm



Homework: What are the differences between Dijkstra's algorithm for SPT and Prim's algorithm for **minimum spanning tree - MST**?
Find the smallest example with a different SPT and MST.

Exploitation of Additional Information

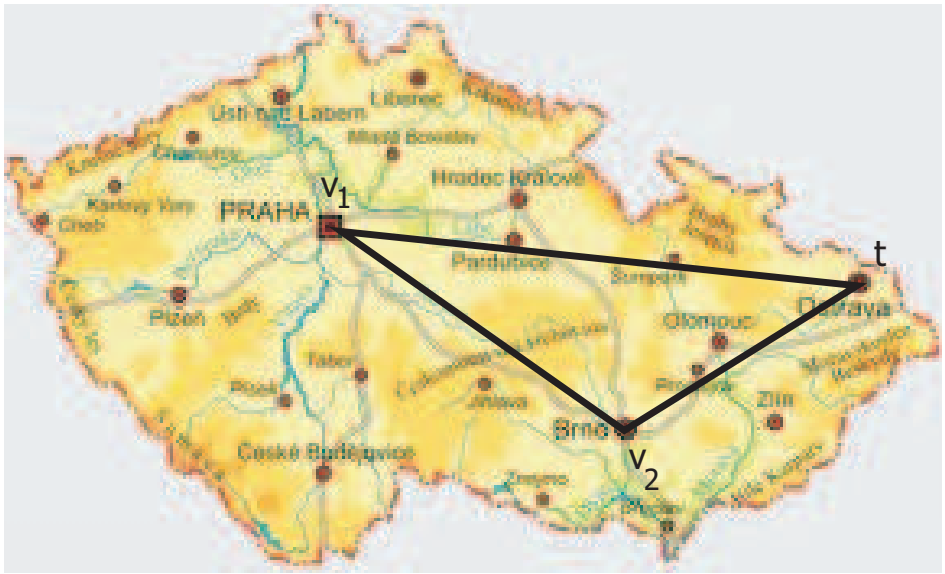
If we are interested in finding the shortest path from s to **just one node** t, Dijkstra's algorithm can be terminated when we remove t from R.
To accelerate the computation time we can add a **heuristic to Dijkstra's algorithm** in order to estimate the length of the remaining path. This is idea of A* algorithm which is generalization of Dijkstra's algorithm that cuts down on the size of the subgraph that must be explored.

- At the algorithm start, we set $h(v, t)$ for every $v \in V(G)$ such that it represents the **lower bound on distance from** v **to** t.
- For arbitrary nodes $v_1, v_2 \in V(G)$ inequality $c(v_1, v_2) \geq h(v_1, t) - h(v_2, t)$ must be valid.
- during Dijkstra's algorithm we choose $v \in V(G) \setminus R$ such that $l(v) = \min_{w \in V(G) \setminus R} \{l(w) + h(w, t)\}$

This algorithm is faster, since it is stopped, whenever the path to t is found and since a high value of h in some vertex excludes this vertex from the choice and arcs leaving this vertex are not explored. Unfortunately, in the the extreme case this algorithm has to go through all vertices as well.

Exploitation of Additional Information - Example

- Suppose the nodes are places in the two dimensional plane with coordinates $[x_v, y_v]$ and arc lengths equal to Euclidean distances between the points. The Euclidean distance from v to t is equal to $[(x_t - x_v)^2 + (y_t - y_v)^2]^{1/2}$, and it can be used as lower bound $h(v, t)$.
- Euclidean distances satisfy triangular inequality, i.e. $h(v_1, v_2) + h(v_2, t) \geq h(v_1, t)$. Therefore $c(v_1, v_2) \geq h(v_1, v_2) \geq h(v_1, t) - h(v_2, t)$



Time Complexity of Dijkstra's Algorithm

- Fastest known algorithm for **SPT** without negative edges
- Time complexity is $O(n^2)$, or $O(m + nlogn)$ using priority queue

Algorithms with linear time complexity for specific problems:

- Planar Graphs - Henzinger [1997]
- Undirected graphs with integer nonnegatice weights - Thorup [1999]

Example SPT.automaton.water: Measurement of Water Level [2]

You are on the bank of the lake and you have one 3-liters bottle and one 5-liters bottle. Both bottles are empty and your task is to have exactly 4 liters in the bigger bottle. You have no other equipment to measure the water level.

a) Represent the problem by the graph.

b) Set-up suitable weights and formulate the shortest path problem to find

- solution with a minimum number of manipulations,
- solution with a minimum amount of manipulated water,
- solution with a minimum amount of water poured back in the lake.

Homework c) During some manipulations you have to be very careful - for example when one bottle is filled completely but the other one does not get empty. Find the solution which minimizes a number of such manipulations.

Homework d) Is it possible to have 5 liters while using 4-liters and 6-liters bottles?

Example SPT.automaton.bridge: Bridge and Torch Problem

Homework - Formulate a shortest path problem:
Four people come to a river in the night. There is a narrow bridge, but it can only hold two people at a time. They have one torch and, because it's night, the torch has to be used when crossing the bridge. First person can cross the bridge in 1 minute, second person in 2 minutes, third person in 5 minutes, and last person in 9 minutes. When two people cross the bridge together, they must move at the slower person's pace. The question is, can they all get across the bridge in 16 minutes or less?

Example SPT.dioid.reliability: Maximum Reliability Path Problem [1]

In the communication network we associate a reliability $p(i,j)$ with every arc from i to j. We assume the failures of links to be unrelated. The reliability of a directed path Q from s to t is the product of the reliability of the arcs in the path. Find the most reliable connection from s to t.

- a) Show that we can reduce the maximum reliability path problem to a shortest path problem.
- b) Suppose you ate not allowed to make reduction. Specify $O(n^2)$ algorithm for solving the maximum reliability path problem.
- c) Will your algorithms in parts a) and b) work if some of the $p(i,j)$ coefficients are strictly greater than 1?

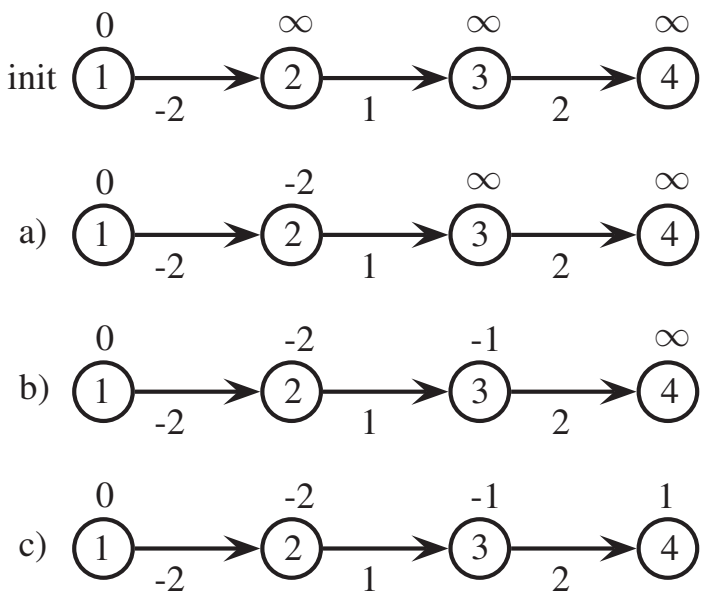
Bellman-Ford Algorithm [1958] (Moore [1959])

Input: directed graph G without a negative circuit
weights $c : E(G) \to \mathbb{R}$ and node $s \in V(G)$.
Output: vectors l and p. For all $v \in V(G)$, $l(v)$ is the length of the shortest path from s and $p(v)$ is the last but one node. If v is not reachable from s, then $l(v) = \infty$ and $p(v)$ is undefined.

$l(s) := 0$; $l(v) := \infty$ for $v \neq s$;
for $i := 1$ **to** $n - 1$ **do**
  for for every edge of graph $(v, w) \in E(G)$ **do**
    if $l(w) > l(v) + c(v, w)$ **then**
      $l(w) := l(v) + c(v, w)$; $p(w) := v$;
    end
  end
end

Bellman-Ford Algorithm - Example

This example illustrates iterations of the Bellman-Ford algorithm finding shortest paths tree from node 1.



Assuming the edges to be processed in the internal loop

- in the worst order, from right to left when only step a) is executed in the first iteration of the external loop, it requires 3 iterations of the external loop to obtain SPT
- in the best order, from left to right when steps a)b)c) are executed in the first iteration of the external loop, it requires only 1 iteration of the external loop to obtain SPT

Correctness of Bellman-Ford algorithm

We will base our reasoning on the following theorem:

Theorem

Let

- $l^k(w)$ is the $l(w)$ label after k iterations of the external loop
- P is the shortest s-w-path with at most k edges and let (v, w) be the last edge of this path
- $c(E(P))$ is the length of path P

Then $l^k(w) \leq c(E(P))$.

Note: the $l^k(w) \leq c(E(P))$ is inequality, since the $l^k(w)$ is the length of the path which might go over more that k edges (see example with four nodes in the chain).

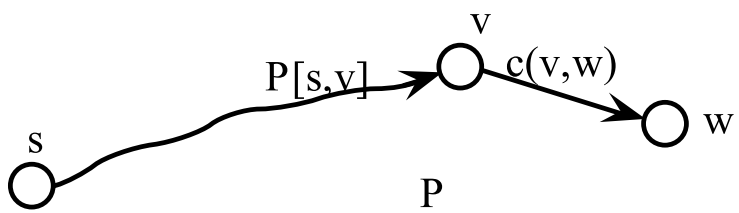
Correctness of Bellman-Ford algorithm - Proof

Prove of the theorem by induction:

- By the induction hypothesis ^{IH} we have $I^{k-1}(v) \leq c(E(P[s,v]))$ after $k-1$ iterations of external loop.
- By Bellman's Principle of Optimality ^{BPO} path segment $P[s,v]$ must be a shortest s-v-path with at most $k-1$ edges.
- In the k-th iteration of the algorithm ^{ALG} edge (v,w) is examined as well.

Finally:

$$I^k(w) \overset{ALG}{\leq} I^{k-1}(v) + c((v,w)) \overset{IH}{\leq} c(E(P[s,v])) + c((v,w)) \overset{BPO}{=} c(E(P)).$$



Since no path has more than $n-1$ edges, the above theorem implies the correctness of the algorithm.

Time Complexity of Bellman-Ford Algorithm and Detection of Negative Circuits

- Best known algorithm for **SPT** without negative circuits.
- Time complexity is $O(nm)$.
- Note that every path consists at most of $n-1$ edges and there exists an edge incident to every reachable node t.
- The Bellman-Ford algorithm **can detect negative circuit** that is reachable from vertex s while checking triangular inequality for resulting l. There may be negative cycles not reachable from s, in such case add node s' and connect it to all nodes v with $c(s,v) = 0$.

for for every edge of graph $(v,t) \in E(G)$ **do**
  if $l(t) > l(v) + c(v,t)$ **then**
    error "Graph contains a negative-weight circuit"
  end
end

However there are better methods for negative circuit detection [Cherkassky&Goldberg 1999].

Example SPT.negative: Truck Journey [2]

Let us assume a truck and n European cities with trailers.

- For each couple of cities (i, j), we know $c(i, j)$, the cost of the truck transport form city i to city j.
- For some couple of cities (i, j) there are (infinitely many) trailers to be transported form city i to city j and the revenue for one trailer is $d(i, j)$.

Our task is to find the track journey from city s to city t and to maximize the profit regardless of the time.

Example SPT.dioid.negative: Investment Opportunities [1]

Mr. Dow Jones, 50 years old, wishes to place his Individual Retirement Account funds in various investment opportunities so that at the age of 65 years, when he withdraws the funds, he has accrued maximum possible amount of money. Assume that Mr. Jones knows the investment alternatives for the next 15 years: their maturity (in years) and appreciation they offer. How would you formulate this investment problem as a shortest path problem, assuming that at any point in time, Mr. Jones invests all his funds in a single investment alternative.

Floyd Algorithm [1962] (Warshall [1962])

Input: a digraph G free of negative circuits and weights $c : E(G) \to \mathbb{R}$.
Output: matrices l and p, where l_{ij} is the length of the shortest path from i to j, p_{ij} is the last but one node on such a path (if it exists).

$l_{ij} := c((i,j))$ for all $(i,j) \in E(G)$;
$l_{ij} := \infty$ for all $(i,j) \notin E(G)$ where $i \neq j$;
$l_{ii} := 0$ for all i;
$p_{ij} := i$ for all (i,j);
for $k := 1$ **to** n **do** `// for all k check if` l_{ij} improves
  for $i := 1$ **to** n **do**
    for $j := 1$ **to** n **do**
      if $l_{ij} > l_{ik} + l_{kj}$ **then**
        $l_{ij} := l_{ik} + l_{kj}$; $p_{ij} := p_{kj}$;
      end
    end
  end
end

Floyd Algorithm

- initialize matrix l^0 by the weights of the edges
- computes the sequence of matrices $l^0, l^1, l^2, \cdots, l^k, \cdots, l^n$ where:

  l_{ij}^k is the length of the shortest path from i to j such that with the exception of i, j it goes only through nodes from the set $\{1, 2, \cdots k\}$
- one can easily compute matrix l^k from matrix l^{k-1}:

$$l_{ij}^k = min\{l_{ij}^{k-1}, l_{ik}^{k-1} + l_{kj}^{k-1}\}$$

Floyd Algorithm - Complexity and Properties

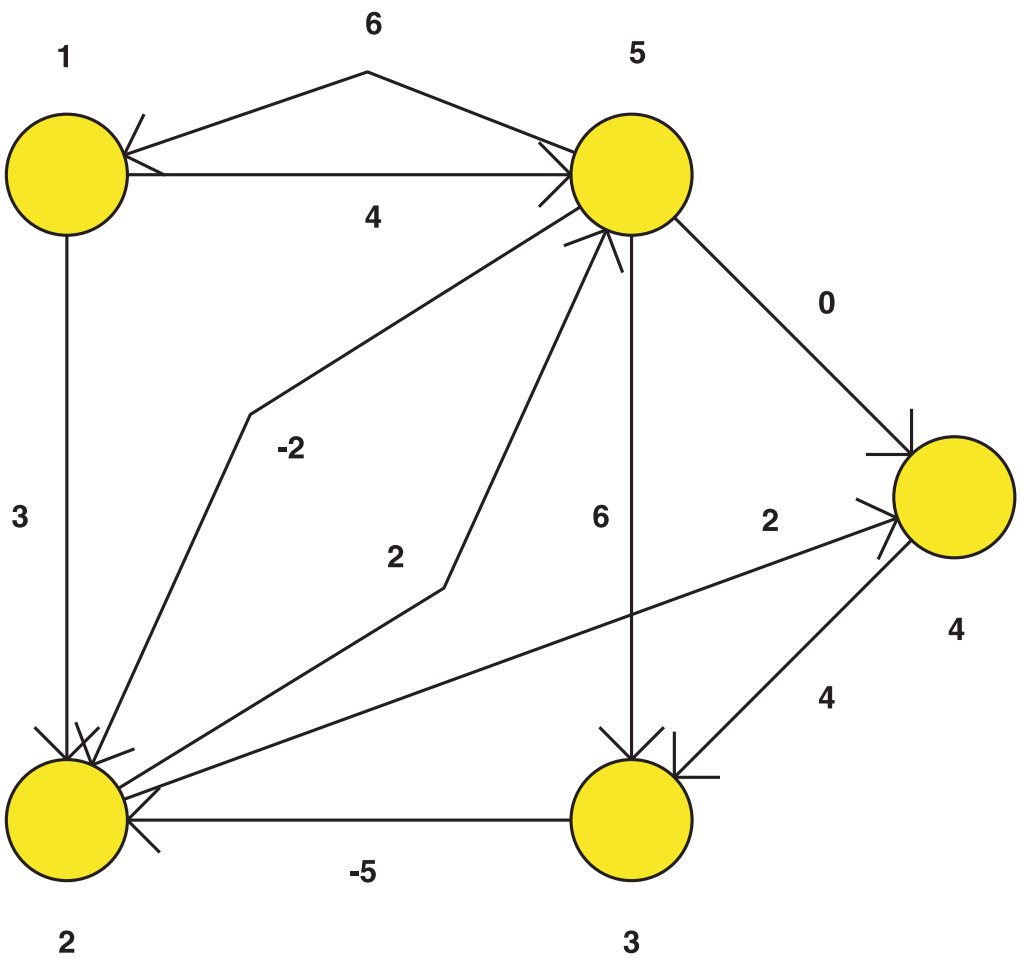
The best known algorithm for **All Pairs Shortest Path** problem without negative circuits.

- Time complexity $O(n^3)$.
- A graph contains a **negative circuit** iff (i.e. if and only if) there exists i such that $l_{ii} < 0$.
- By a small modification of the algorithm, where l_{ii}^0 is set to ∞, one can find the nonnegative **minimal weight circle** - see the diagonal in the next example.

Johnson's Algorithm is better suited for the sparse graphs

- uses Dijkstra and Bellman-Ford
- complexity: $O(|V| \cdot |E| \cdot log|V|)$ (in the simplest case)

Floyd Algorithm - Example



$$l^0 = \begin{pmatrix} \infty & 3 & \infty & \infty & 4 \\ \infty & \infty & \infty & 2 & 2 \\ \infty & -5 & \infty & \infty & \infty \\ \infty & \infty & 4 & \infty & \infty \\ 6 & -2 & 6 & 0 & \infty \end{pmatrix}$$

$$l = \begin{pmatrix} 10 & 2 & 8 & 4 & 4 \\ 8 & 0 & 6 & 2 & 2 \\ 3 & -5 & 1 & -3 & -3 \\ 7 & -1 & 4 & 1 & 1 \\ 6 & -2 & 4 & 0 & 0 \end{pmatrix}$$

$$p = \begin{pmatrix} 5 & 5 & 4 & 5 & 1 \\ 5 & 5 & 4 & 2 & 2 \\ 5 & 3 & 4 & 2 & 2 \\ 5 & 3 & 4 & 2 & 2 \\ 5 & 5 & 4 & 5 & 2 \end{pmatrix}$$

Example SPT.matrix.fire: Location of Fire Station [2]

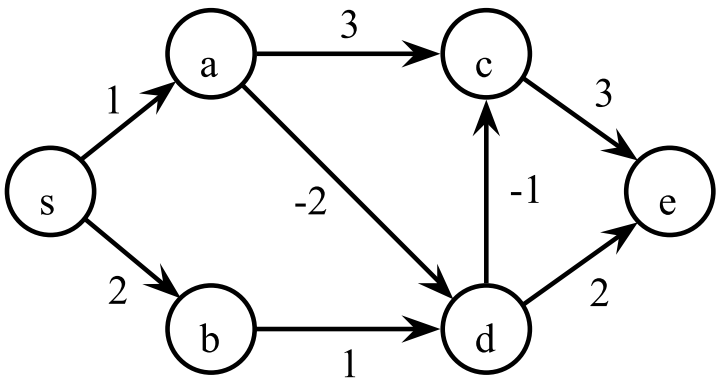
Formulate a shortest path problem:
Consider a road system in the city.
a) We are looking for the best location of the fire station while minimizing its distance form the most distant place.
Homework b) How the problem changes (gets more difficult) when we are looking for the best location of two fire stations?
Homework c) How the problem changes (gets more difficult) when the maximum allowed distance is given and we are looking for the minimum number of stations.

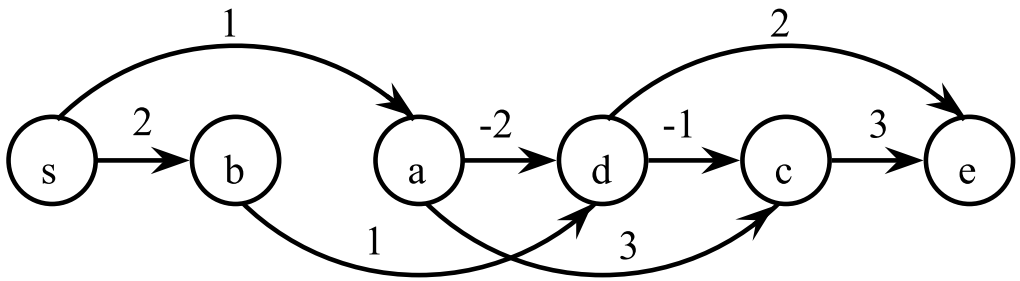
Example SPT.matrix.warehouse: Warehouse Location

Formulate a shortest path problem:
Consider a road system in the region. We are looking for the best location of the warehouse which supplies n customers consuming $q_1, \ldots, q_n$ units of goods per week. There is a suitable place for warehouse nearby each of the customers. Each customer is served by a separate car and the transport cost is a product (i.e. multiplication) of the distance and transported volume. Objective is to minimize the weekly transport costs.

Shortest Paths in Directed Acyclic Graphs (DAGs)

Definition: A **topological order** of G is an order of the vertices $V(G) = v_1, \ldots, v_n$ such that for each edge $(v_i, v_j) \in E(G)$ we have $i < j$.
Proposition: A directed graph has a topological order if and only if it is acyclic (see the proof in [3]).
Consequence: DAG vertices can be arranged on a line so that all edges go from left to right.





Observation: shortest path from s to v_i cannot use any node from $v_{i+1}, \ldots, v_n$, therefore we can find the shortest paths in topological order

Algorithm for DAGs

May be seen as simplified version of Bellman-Ford algorithm.

Input: directed acyclic graph G with topologically ordered vertices $v_1, \ldots, v_n$, weights $c : E(G) \to \mathbb{R}$.

Output: vectors l and p. For all $i = 1 \ldots n$, $l(v_i)$ is the length of the shortest path from v_1 and $p(v_i)$ is the last but one node. If v_i is not reachable from v_1, then $l(v_i) = \infty$ and $p(v_i)$ is undefined.

$l(v_1) := 0$; $l(v_i) := \infty$ for $i = 2 \ldots n$;
for $i := 2$ **to** n **do**
  for for every edge of graph $(v_j, v_i) \in E(G)$ **do**
    if $l(v_i) > l(v_j) + c(v_j, w)$ **then**
      $l(v_i) := l(v_j) + c(v_j, v_i)$; $p(v_i) := v_j$;
    end
  end
end

Correctness: induction on i and observation on previous slide.

Time complexity $O(|V| + |E|)$

Dynamic Programming Perspective

Observation on SPT algorithm for DAGs which:

- solves a collection of subproblems, e.g. $l(d) = min\{l(a) - 2, l(b) + 1\}$
- starts with simplest one, i.e. $l(s) = 0$
- proceeds with larger subproblems along the topological order

This is a general technique called Dynamic Programming, which requires two key attributes: **optimal substructure** and **overlapping subproblems**.

In the case of the SPT algorithm for DAG the solution space (showing dependencies between subproblems) may be represented by the same DAG.

Graphical representation of the solution space in the case of Bellman-Ford algorithm or Floyd algorithm is different from the input graph G, but it has optimal substructure and overlapping subproblems as well.

Dynamic Programming Perspective

State space is represented by a graph. It is obvious in the case of SPT, but in other optimization problems it requires integer valued variables (e.g. weights or costs of items in the **Knapsack problem**).
Optimal substructure - the solution to a given optimization problem can be obtained by the **combination of optimal solutions to its subproblems**. Such optimal substructures are usually described by means of **recursion** (examples are **Bellman equation** in Bellman-Ford algorithm and "**shortest path consists of shortest paths**" in Floyd algorithm).

Overlapping subproblems - computed **solutions to subproblems are stored** so that these don't have to recomputed. So Dynamic Programming is not useful when there are no common (overlapping) subproblems because there is no point storing the solutions if they are not needed again. One can observe so called **diamonds** in the in the solution space, in contrast to the solution space of Branch and Bound algorithm, which can be represented by the tree.

Shortest Path - Conclusion

- An easy optimization problem with a lot of practical applications
  - OSPF (Open Shortest Path First) is a widely used protocol for Internet routing that uses Dijkstra's algorithm.
  - RIP (Routing Information Protocol) is another routing protocol based on the Bellman-Ford algorithm.
  - looking for shortest/cheapest/fastest route in the map
- Basic routine for many optimization problems
  - scheduling with dependencies
  - ...

References


Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin.
Network Flows: Theory, Algorithms, and Applications.
Prentice Hall, 1993.


Jiří Demel.
Grafy a jejich aplikace.
Academia, 2002.


B. H. Korte and Jens Vygen.
Combinatorial Optimization: Theory and Algorithms.
Springer, fourth edition, 2008.