Combinatorial Optimization
Lab 09: Shortest paths

Industrial Informatics Department
industrialinformatics.fel.cvut.cz

April 10, 2026

Abstract

In this handout, you will revise your knowledge about the shortest path problem, its algorithms and complexity. Next, we will take a look on the application concerning function approximation.

Part 1: Shortest paths

- What is the problem of the shortest paths?

  For graph $G = (V, E)$ (either oriented or not) and a pair of vertices (v_s, v_t), we want to find path $P = (v_s, \ldots, v_t)$ such that its cost is minimized. Path is a sequence of vertices $(v_1, v_2, \ldots, v_n)$, such that v_i and v_{i+1} are adjacent vertices (there exists an edge between them). Cost of the path is defined as the sum of the costs of its edges.

- What is the difference between path (cesta) and walk (sled)?

  Walk – vertices and edges can repeat. Path – vertices cannot repeat. (Note: trail (tah) is walk, where no edges repeat)

- Which algorithms can be used to solve the shortest path problem? What are their (dis)advantages?

  - Dijkstra (one-to-all) – only non-negative weights; $\mathcal{O}(n^2)$ or $\mathcal{O}(m + n \log(n))$ – depending on the implementation
  - A* (one-to-one) – Djisktra with heuristic (reduce the size of the search space)
  - Bellman–Ford (one-to-all) – edge weights may be negative, but no negative cycles; can detect negative cycles; complexity $\mathcal{O}(m \cdot n)$.
  - Floyd-Warshall (all-to-all) – no negative cycles; complexity $\mathcal{O}(n^3)$

  All of these algorithms are polynomial – i.e., they solve only some subset of all shortest-paths problems. Specifically, only the problems without negative cycles can be solved like that (the algorithms do not distinguish between paths and walks (edge progressions)). Note that if there are negative cycles, the shortest edge progression does not need to exist! Shortest path problem is $\mathcal{NP}$-hard in general.

- Polynomial reduction of the Hamiltonian path (HP) to the shortest paths.

  Just transform the graph by adding new source and target connected to all of the original vertices by edges with zero cost, and set the costs of the original edges to (-1), see Figure 1 (b). Now we try to find the minimal cost path from s' to t'. If the cost is $-(|V| - 1)$, then the HP in the original graph exists, otherwise it does not exist. Note that Hamiltonian path problem for acyclic graph is solvable in polynomial time (do the topological sort and check if there is an edge between each two adjacent nodes).

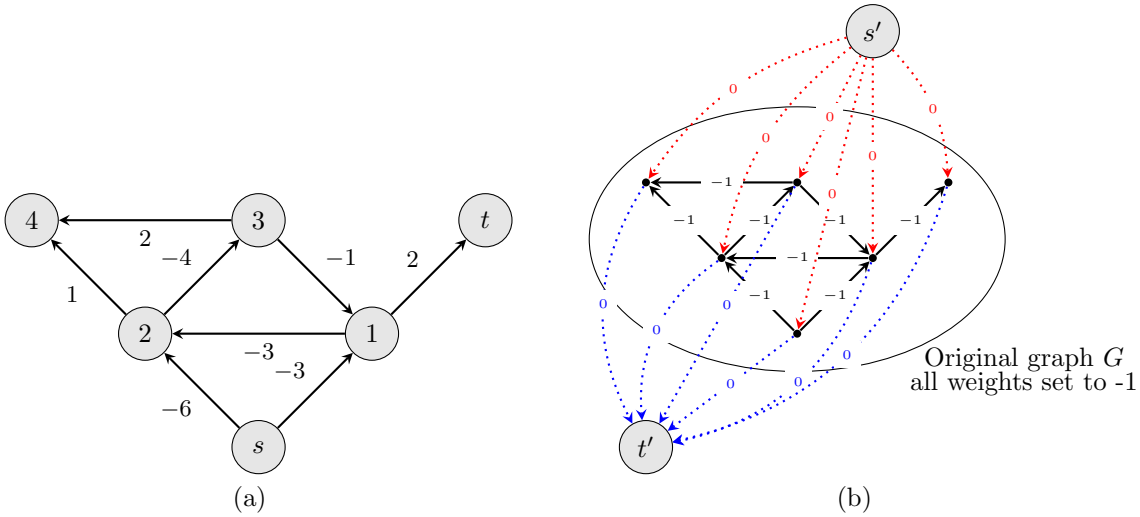

Figure 1: Example of graph with negative cycles (a); reduction of HP to the shortest path problem (b)

Part 2: Approximation of a function

Let us have a set of data points $S = \{(x_i, f(x_i)) \mid i \in \{1, 2, \dots n\}\}$ representing values of function f. We assume that n is large. To represent function f, we want to select subset $S' \subseteq S$, which will approximate the original function. The approximation is done by linear segments (between each two adjacent selected data points). Of course that there will be some **approximation error**, which should be minimized.

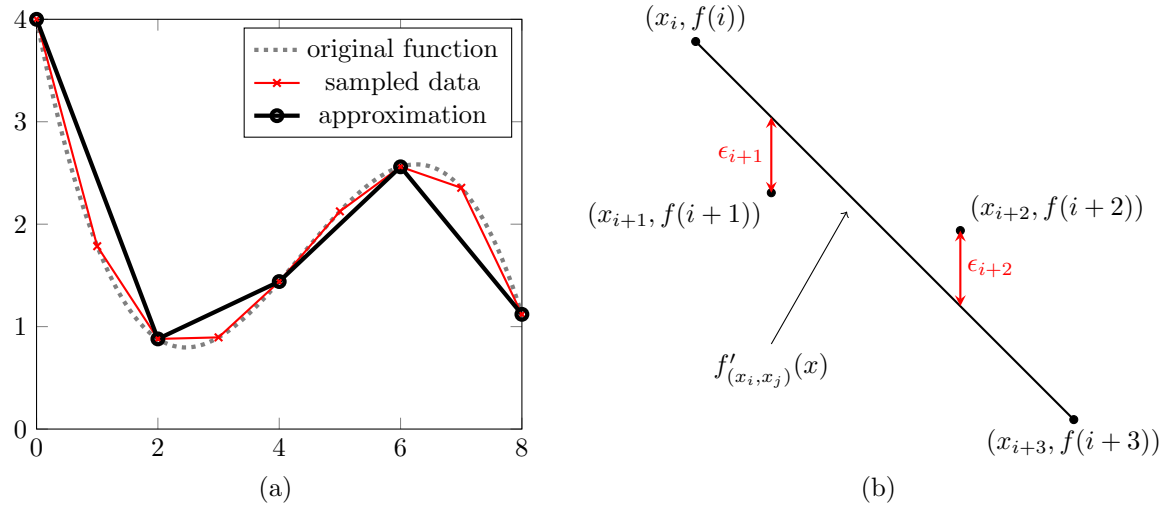

Figure 2: Original function and approximation (a); approximation error between two selected points (b)

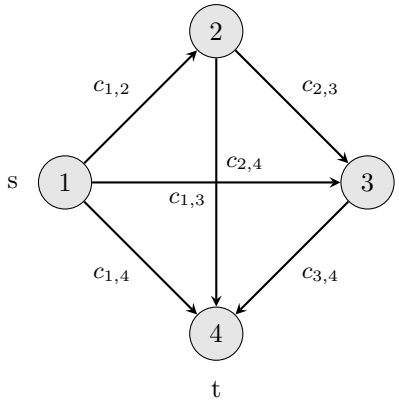

Figure 3: Example of graph G for 4 data points

The error (cost) of approximation between selected points x_i and x_j can be written as

$$c_{i,j} = \sum_{k=i+1}^{j-1} \epsilon_k^2 = \sum_{k=i+1}^{j-1} [f(x_k) - f'_{(x_i,x_j)}(x_k)]^2,$$

where function $f'_{(x_i,x_j)}(x)$ represents the line segment between points $(x_i, f(i))$ and $(x_j, f(j))$.

And now the funny part: Approximation of function f can be in fact transformed to a shortest path problem. We can create graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ and $E = \{(i, j) \mid i < j \wedge i, j \in V\}$. Vertices correspond to the individual data points, and the oriented edges link each pair of different data points. Cost of the edge $c_{i,j}$ corresponds to the approximation error, i.e., if vertices i and j are on the shortest path in the graph from vertex 1 to vertex n, then part of the approximation is the line segment between points $(x_i, f(i))$ and $(x_j, f(j))$ and the corresponding cost is the approximation error between these two points.

What is the problem? Now, what would happen if we started the algorithm? What would be the optimal solution?

The optimal solution would be 0. Why? Because we do not penalize the number of selected points! Therefore we could choose every point – giving zero error. Therefore we need to modify the costs. One possible modification is the following one:

$$c_{i,j} = \alpha + \beta \cdot \sum_{k=i+1}^{j-1} \epsilon_k^2,$$

where α and β are parameters. By modifying these parameters we penalize the number of edges versus the approximation error. We have already seen that if α was zero, the shortest path would contain all of the edges. On the other hand, if $\alpha >> \beta$, then there would only be one edge (from the first data point to the last one).

Interesting application: Same ideas can be used in the field of vector images. However, the error should be modified. What is some reasonable error function for 2D vector images?

We can just use (perpendicular) distance of the point from the line segment. The formula is written in the handout.

Now, see the Czech republic example. You can implement the approximation of the Czech republic. An example of two-dimensional approximation, where the frontiers of the Czech Republic are input data, is depicted in Figure 4.

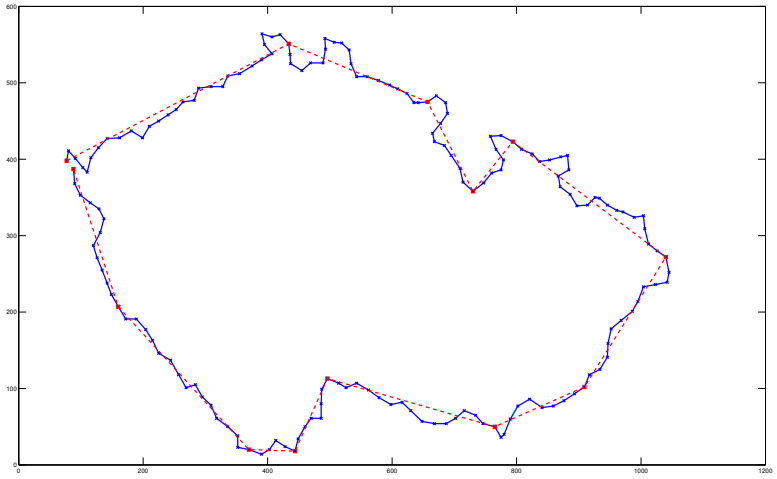

Figure 4: Entered points (blue) and selected points (red).