DAST 2004

Tirgul 13

Single-Source-Shortest-Paths

DAST 2004

Unweighted Graphs

•Wishful Thinking – you decide to go to work on your sun-tan in 'Hatzuk' beach in Tel-Aviv. Therefore, you take your swimming suit and Tel-Aviv's bus trails map , and go on bus 405 to the central station of Tel-Aviv. The bus ride inside the city costs 1 nis per station.

• How will you find the cheapest way from the central station to the beach?

DAST 2004

Unweighted Graphs - cont.

• **BFS** finds shortest paths from a single source (i.e – the central station) in an unweighted graph.

• How much will you pay?

• n nis, when n is the number of stations you passed in your way.

•Can you think of an algorithm that finds a single shortest **path**, and always works better then BFS?

•No such algorithm is known.

Weighted Graphs

- Now assume that you pay for the bus ride between stations according to the distance between the stations. That is – every 'edge' is the bus trails map has a different price (= weight).
- Total payment = sum over the costs between the stations on the way.
- Will BFS work?
- No – BFS counts the number of edges on the path, but does not refer to the edges weights.

Other Versions of Shortest Paths

- If we know how to find shortest paths from a single source, we can also find:
- A shortest path between a pair of vertices
- The shortest paths from all vertices to a single source. How?
- By reversing the direction of the edges in the graph
- Shortest paths between every pair of vertices. How?
- By running the sssp from every vertex (there are more efficient solutions)

Observations

Observation 1:When can negative weights become a problem?

- When there is a circle with negative weight, reduce the weight of the path by repeating the circle over and over.
- Solution: either require non-negative weights, or identify and report circles with negative weights

Observations – cont.

Observation 2: Let u ~ y ~ z ~ v be the shortest path between u and v. Is y ~ z optimal, too?

- Yes! If there was a shorter path between y and z, we could use it to shorten the path between u and v ➔ contradiction
- This property suggests that we can use greedy algorithms to find the shortest path

Relaxation

The Idea: Build a shortest paths tree rooted in S.

For every vertex, *v*, keep a value, *d[v]*, of the shortest path from s that is currently known.

The general algorithm scheme:

- Initialization: $d[v] = \infty$, $d[s] = 0$
- In every iteration of the algorithm we check if we can do **relaxation** – that is, find a shorter path from s to a vertex *v* then the path currently known.

We will learn two algorithms:

- Dijkstra – all weights are non-negative
- Belman-Ford – identifies circles with negative weight

Dijkstra

- The idea: Maintain a set of vertices whose final shortest path weights from *s* have already been determined
- How will we choose the next vertex to add to the set?
- We will take the vertex *u* with the minimal *d* value. This is also the ‘real’ value of the shortest path.
- What relaxation can be done?
- We can check all the edges leaving *u*.

Dijkstra Run Time

- How can we keep the vertices, so we can easily find the next vertex to insert?
- We need to extract the minimal *d* value in each iteration, so a binary heap is a good choice.

- Run time alalysis:
  - Build heap takes O(V).
  - Extract-Min O(logV).
  - Altogether O(VlogV).
  - Going over the adjacent list O(E).
  - Relaxation of values in the heap O(logV).
  - Altogether O(ElogV).

- Total run-time compexity: O(ElogV+VlogV) = O(ElogV)

Bellman-Ford's Algorithm

- Now we want to identify negative circles.
- Assume that every iteration we do relaxation from all the edges. What edges might be relaxed on iteration i?
- The edges that have a path with I edges from s (shorter paths – updated in previous iterations)
- What is the maximum number of edges in any shortest path from s?

Bellman-Ford's Algorithm

- So – how many iterations are needed if there are no negative circles?
- |V| - 1. After the |V| - 1 iteration, all *d* values will be the lengths of the shortest paths.
- And how can we identify negative circles?
- By running the |V| iteration – if we can find a relaxation, then there is a negative circle in the graph

Bellman-Ford's Algorithm

Initialize(G, s)
for $i \leftarrow 1$ to $|V| - 1$
  for each edge $(u, v) \in E[G]$
    do Relax(u, v, w)
for each edge $(u, v) \in E[G]$
  if $d[v] > d[u] + w(u, v)$ return false
return true

Bellman-Ford Run Time

- The algorithm run time is:
  - Goes over v-1 vertexes, O(V)
  - For each vertex relaxation over E, O(E)
  - Altogether O(VE)

Application of Bellman-Ford

- Linear programming problems – find an optimal value (min / max) of a linear function of n variables, with linear contraints.
- A special case: Set of difference constraints

$$
\begin{aligned}
x_1 - x_2 &\le 0 \\
x_1 - x_5 &\le -1 \\
x_2 - x_5 &\le 1 \\
x_3 - x_1 &\le 5 \\
x_4 - x_1 &\le 4 \\
x_4 - x_3 &\le -1 \\
x_5 - x_3 &\le -3 \\
x_5 - x_4 &\le -3
\end{aligned}
$$

Application of Bellman-Ford

- There are many uses for a set of difference constraints, for instance:
  - The variables can represent the times of different events
  - The inequalities are the constraints over there synchronization.
- The set of linear inequalities can also be expressed in matrix notation:

$$\boldsymbol{A} \cdot \boldsymbol{x} \le \boldsymbol{b}$$

Application of Bellman-Ford

$$\underbrace{\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix}}_{\boldsymbol{A}} \cdot \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}}_{\boldsymbol{x}} \le \underbrace{\begin{pmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{pmatrix}}_{\boldsymbol{b}}$$

Application of Bellman-Ford

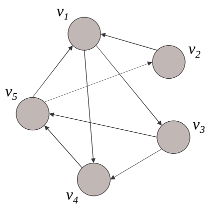
- What is the connection between the Bellman-Ford algorithm and a set of linear inequalities?
- We can interpret the problem as a directed graph.
- The graph is called the *constraint graph* of the problem
- After constructing the graph, we could use the Bellman-Ford algorithm.
- The result of the Bellman-Ford algorithm is the vector *x* that solves the set of inequalities.

Application of Bellman-Ford

- Building the *constraint graph* out of matrix A:
  - Each variable represents a vertex (node)
  - Each constraint represents an edge
  - If edge i goes **out of** vertex j than $A[i,j] = -1$
  - If edge i goes **into** vertex j than $A[i,j] = 1$
  - Otherwise $A[i,j] = 0$
  - Each row contains a single '1', a single '-1' and zeros.

Application of Bellman-Ford

- The problem is represented by the graph:



Application of Bellman-Ford

- How can we extend the *constraint graph to a* single-source-shortest-paths problem?
- By adding vertex v_0 that directs at all the other vertices.
- Weight all edges from v_0 as zero.
- The weight of the other edges is determined by the inequality constraints.

DAST 2004

Application of Bellman-Ford

- Formally:
- Each node v_i corresponds to a variable x_i in the original problem, and an extra node - v_0 (will be s).
  $V = \{v_0, v_1, \ldots, v_n\}$
- Each edge is a constraint, except for edges (v_0, v_i) that were added.

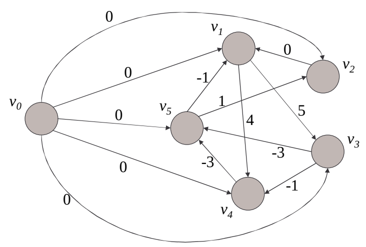
$$E = \{(v_i, v_j) \mid x_j - x_i \le b \text{ is a constraint }\} \cup \{(v_0, v_i) \mid i = 1..n\}$$

- Assign weights:
  $w(v_0, v_i) = 0$ for $i = 1 \ldots n$
  $w(v_i, v_j) = b_k$ if $x_i - x_j \le b_k$ is a constraint

DAST 2004

Application of Bellman-Ford

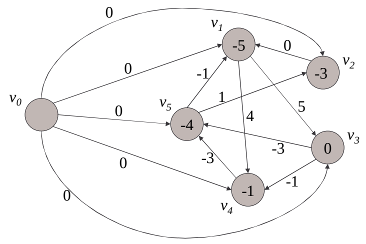
- The extended constraint graph:



DAST 2004

Application of Bellman-Ford

- The solution:



Application of Bellman-Ford

- The Bellman-Ford solution $\delta(v_0, v_i)$ for the extended constraint graph is a set of values which meets the constraints.
- Formally:
  $$x_i := \delta(v_0, v_i)$$
- Why is this correct?

Application of Bellman-Ford

- Because $\forall j, i \quad j \neq i$

$$\delta(v_0, v_j) \le \delta(v_0, v_i) + w(v_i, v_j)$$
$$\Rightarrow \delta(v_0, v_j) - \delta(v_0, v_i) \le w(v_i, v_j)$$
$$\Rightarrow x_j - x_i \le b_k \quad (\text{for some } k)$$

Application of Bellman-Ford

- If there is a negative cycle reachable from v_0 – there are no feasible solutions:
- Suppose the cycle is $\langle v_1, v_2, \dots, v_k \rangle$ where $v_k = v_1$
- v_0 cannot be on it (it has no incoming edges)
- This cycle corresponds to :

$$x_2 - x_1 \le b_{k_{2,1}} = w(v_1, v_2)$$
$$x_3 - x_2 \le b_{k_{3,2}} = w(v_2, v_3)$$
$$\vdots$$
$$x_1 - x_{k-1} \le b_{k_{1,k-1}} = w(v_1, v_{k-1})$$

Application of Bellman-Ford

- The left side sums to 0.

$$\sum_{i=1}^{k-1}(x_{i+1} - x_i) = x_k - x_1 = 0$$

- The right side sums to the cycle's weight *w(c)*
- We get $0 \leq w(c)$
- But we assumed the cycle was negative…