CS 4301 — Special Topics in Computer Science: Optimization, Homework 4

https://iniyanjoseph.com | iniyanijoseph@gmail.com

$$\max \pi(t) - z * \pi(s)$$

$$\pi(v) - \pi(u) \leq w(uv);\ \forall (u,v) \in A$$

Increasing d by a certain amount made the arc $u \to v$ tight, which wouldn't be possible if $u, v \in \Pi$ or $u, v \notin \Pi$, since increasing d wouldn't change $\pi(v) - \pi(u)$. Additionally, if $u \notin \Pi$ and $v \in \Pi$, then increasing d would decrease the value of $\pi(v) - \pi(u)$, which couldn't make the condition $\pi(v) - \pi(u) \leq w(uv)$ tight. Therefore, increasing d will only make an edge go tight if $u \in \Pi$ and $v \notin \Pi$. Because $u \in \Pi, v \notin \Pi$, in every iteration of the loop, we add a vertex v to Π. This is a shortest path. This is obviously true in the beginning, because the edge that goes tight the cheapest (s, v) for a vertex v. Because there is a $u \in \Pi$ to which the vertex newly added v is connected, there is a path to the newly added vertex, where the path is a subset of Π, the statement holds true in every iteration of the loop. It is the cheapest path because if there was a cheaper edge connecting a vertex in Π to a vertex outside of Π, those edges would go tight first. Thus, the edge which goes tight is an edge of an shortest s, v path which we are constructing. Once t is added to the path, that means that Π is a tree, containing a minimum s, t path, and the algorithm can terminate. (Indeed, it also contains the shortest path from s to any other vertex $v \in V \setminus \{s\}$).

The quantity $\pi(v)$ is equal to the sum of the weights of the arcs on the (s, t) path P, since all these arcs are tight. In our algorithm, we continuously add the vertex for which we have a "cheapest" path to that vertex in a BFS-style manner. This is Dijkstra's Algorithm.

Let $U = \{v \in V \mid v_1 v_2 \notin M\}$. Label an edge, $uv \in P$, red if $u_1 v_1 \in M$, and blue if $u_2 v_2 \in M$. No edge can be both red and blue because otherwise, a lesser weight matching could be made with the edges $u_1 u_2$ and $v_1 v_2$.

Because every vertex must be matched, s, t must be matched to some vertex in G_1. This creates two vertices in G_2 which cannot be matched using an edge connected to G_1. These vertices must then match to two vertices in G_2 by the definition of a perfect matching. This creates another two edges in G_1 which we call s' and t'. This holds inductively until every vertex is matched with no loops in the projection. Because no edge $u_1 v_1, u_2 v_2$ is taken twice and P contains no loops, it is an odd length path in G.

The number of edges in P must be odd (and thus the number of vertices even), since otherwise it would be possible to form a cheaper matching using a $u_1 u_2$ for at least one vertex u not matched the matching.

P contains a path from s, t. Additionally, since each vertex copy can be matched to at most one edge, each vertex in the projection has a degree of at most 2. Let C be a set of vertices that contain a cycle in P. Each vertex's quota of two edges is accounted for by the cycle, so C is an independent set from the rest of the vertices. Additionally, since C accounts for C edges in the matching, each of which are of positive weight, we could replace the edges between C with the edges $c_1 c_2$ for each $c \in C$, for a lower cost matching, leading to a contradiction. Therefore, P contains no cycles.

Because the projection P is an s, t path with no cycles, has no repeated edges, and has an odd number of edges, and because the non-loop edges are of minimum weight, P is a path. It is the shortest path, because if there were a cheaper s, t path, there would also be a corresponding cheaper matching that could be formed as follows: Label each edge in the path $1, 2, 3, ..., k$. For each edge with an odd label, (including the edges incident to s and t since the path is of odd length), the edge can be taken in the matching in G_1. For every edge with an even label, the edge can be taken in the matching in G_2. For all other vertices u, a $u_1 u_2$ edge can be taken in the matching.

The new graph is of strongly polynomial size and takes strongly polynomial time to create. Computing a perfect matching is strongly polynomial, thus the algorithm takes strongly polynomial time to run.