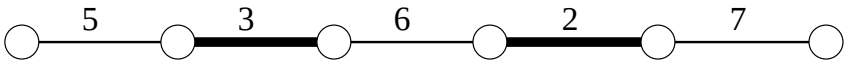
1 Weighted matchings in bipartite graphs

Let $G = (V, E)$ be an undirected graph with integer edge weights $w : E \to \mathbb{Z}$. A *matching* in G is a subset $M \subseteq E$ of edges such that no two edges in M share a common vertex. A matching M is called *matching of maximal cardinality* if there is no matching M' in G with $|M'| > |M|$. The weight $w(M)$ of a matching M is the sum of the weights of all edges in M. The weight $w(p)$ of an augmenting path p in M is the sum of the weights of all edges in $p \setminus M$ minus the sum of the weights of all edges in $p \cap M$. The weight of the following augmenting path is then $5 + 6 + 7 - 3 - 2 = 13$.



With this defintion, the weight of a matching changes by $w(p)$ when inverting an augmenting path p.

We have already seen the Hopcraft-Karp algorithm, which efficiently computes matchings of maximal cardinality in bipartite graphs in time $O(\sqrt{|V|}|E|)$. But often there are different matchings of maximal cardinality, and we do not just want to find any of those, but one with minimal or maximal weight. In the following, we consider the case of finding a matching of maximal cardinality and minimal weight.

1.1 Incremental computation

The basic idea of solving this problem is to compute a matching M_k, which contains exactly k edges and has minimal weight among all those with k edges, for every $k = 1, 2, \ldots$ consecutively. The following theorem shows how M_{k+1} can be computed from M_k.

Theorem 1 *Let M_k be a matching in G with k edges, which has minimal weight among all matchings with exactly k edges. Let p be an augmenting path in G with respect to M_k, which has minimal weight among all augmenting paths with respect to M_k. Then $M_{k+1} := M_k \oplus p$, i.e. the resulting matching after inverting p in M_k, contains exactly $k + 1$ edges and has minimal weight among all matching in G with exactly $k + 1$ edges.*

Proof: Let M' be an arbitrary matching with $k + 1$ edges, which has minimal weight among all those with $k + 1$ edges. Consider the graph $H = (V, M_k \oplus M')$ which only contains edges included in either M_k or M' (but not in both). All vertices in H have a degree of at most 2, and the connected components of H can only be of the following form:

1. alternating (w.r.t. M_k and M') paths or cycles with an even number of edges
2. augmenting paths w.r.t. M_k (i.e. by inverting such a path, $|M_k|$ increases by one, and $|M'|$ decreases by one, respectively)
3. augmenting paths w.r.t. M' (i.e. by inverting such a path $|M_k|$ decreases by one, and M' increases by one, respectively)

The weight of the edges from M_k on an alternating path or a cycle of even length must be equal to the weight of the edges from M', otherwise M_k or M' could be made cheaper (contradiction). Furthermore, the number of augmenting paths w.r.t M_k (2.) must be exactly one more than the number of augmenting paths w.r.t. M' (3.).

If H contains no augmenting path w.r.t M' (3.), we are done: H contains exactly one augmenting path p w.r.t. M_k and by inverting p (or every other augmenting path with minimal weight) we obtain from M_k a matching with $k+1$ edges and minimal weight.

If H contains an augmenting path w.r.t. M' (3.), for every pair of an augmenting path p w.r.t. M_k and an augmenting path q w.r.t. M' the following must hold: the sum of the edge weights in $(p \cup q) \cap M_k$ is equal to the sum of the edge weights in $(p \cup q) \cap M'$. (Otherwise either M_k or M' can be made cheaper by inverting p and q, a contradiction.) Hence, all augmenting paths w.r.t. M_k must have the same weight $c \in \mathbb{Z}$, and all augmenting paths w.r.t. M' must have the same weight $-c$. In this case, it immediately follows that a matching with $k+1$ edges and minimal weight can be computed from M_k by inverting a single augmenting path w.r.t. M_k with weight c, which can also be done with an arbitrary augmenting path with minimal weight. □

The algorithm for finding a matching of maximal cardinality and minimal weight is then the following:

- set $M := \emptyset$
- **while** (there exists augmenting path w.r.t. M) **do**
  search for an augmenting path p with minimal weight;
  invert p and increase M by one edge;
  od
- output M as matching of maximal cardinality and minimal weight

1.2 Computation of an augmenting path with minimal weight

What is still left is how to find an augmenting path with minimal weight in a bipartite graph. So, we want to search all the alternating paths, which go from a free vertex in V_1 to a free vertex in V_2, for one with minimal weight. We observe that each of those paths visits edges in $E \setminus M$ only in the direction from V_1 to V_2. and edges in M only in the direction from V_2 to V_1. Thus, we may consider these edges as directed arcs. Furthermore we can assign costs to these arcs such that the weight of an augmenting path is equal to the sum of the costs on this path: every arc $e \in E \setminus M$ is assigned the cost $c(e) = w(e)$ and every arc $e \in M$ is assigned the cost $c(e) = -w(e)$. Now, if we add two new vertices s and t, connect s to all free vertices in V_1 via arcs with $c(e) = 0$, and connect all free certices in V_2 to t via arcs with $c(e) = 0$, we can find an augmenting path with minimal weight by finding a shortest path from s to t in this extended graph G'. We still need to check how to find such a shortest path from s to t in G'. (There are some examples of such a construction in section 1.3.)

1.2.1 Negative weights and recalibration

Because the arcs in G' can have negative weights we may not be able to use Dijkstra's algorithm. We can use the Bellman-Ford algorithm instead, which computes shortest paths in graph with arbitrary weights (but without cycles of negative length) in time $O(|V| \cdot |E|)$. The overall running time for finding a matching with maximal cardinality and minimal weight is $O(|V|^2|E|)$, because we have to find $|V|/2$ augmenting paths in the worst case.

But by using a clever "recalibration" trick we can change the costs of the arcs to positive values and then use Dijkstra's algorithm after all. Because Dijkstra's algorithm has a worst case running time of $O(|E| + |V| \log |V|)$ by using an implementation with Fibonacci heaps, this yields an overall running time of $O(|V|^2 \log |V| + |V| \cdot |E|)$ for finding a matching of maximal cardinality and minimal weight (among all these).

So, how does such a recalibration work? First, consider an arbitrary function $F : V \to \mathbb{Z}$ and replace the costs $c(u, v)$ of each edge (u, v) in G' by $c'(u, v) := c(u, v)+F(u)-F(v)$. The length of a path from s to t then changes by exactly $F(s)-F(t)$. Because this change does not depend on the path, a shortest path from s to t remains a shortest path even after modifying the costs. If we find a function F, which turns all the values $c'(u, v)$ positive, then we can find an augmenting path with minimal weight by using Dijkstra's algorithm.

1.2.2 Choice of the recalibration function

We need a function F such that all $c'(e)$ are nonnegative after recalibration. One possibility is to choose $F(u)$ equal to the length of a shortest path from s to u: this way, $F(v) \leq F(u) + c(u, v)$ holds for every arc (u, v), yielding $c'(u, v) \geq 0$ directly. The problem with this approach is that we want to find the shortest paths by using the recalibration. The solution to this dilemma is to use the information of the search for an augmenting path with minimal weight from the step before!

If, at the beginning, i.e. before finding the first augmenting path, there exist arcs with negative costs, then we just add a sufficiently large number to all the arc costs to make them positive. (We can do that, because the weights of all matchings of maximal cardinality changes in the same way.) Thus, finding the first augmenting path can be done using Dijkstra's algorithm. The graph G' and the costs of the arcs are set as mentioned above. Then Dijkstra's algorithm is run until the lengths of the shortest paths to all vertices in G' are computed (*single source* problem).

By $c(u, v)$ we denote the current cost of the arc (u, v) in G' and by $d(v)$ we denote the calculated length of a shortest path from s to v. Furthermore, let p be the shortest path in G' from s to t, given by Dijkstra's algorithm. Then the matching algorithm executes the following operations:

- The augmenting path in G corresponding to p is inverted.
- The first and last arcs of p are deleted from G'.
- For all arcs (u, v) in G' not included in p, the costs $c(u, v)$ are replaced by $c(u, v) + d(u) - d(v) \geq 0$.

- All remaining arcs (u, v) of p in G' (i.e. all arcs apart from the first and last one) are inverted (i.e. (u, v) is replaced by (v, u)) and are assigned new costs $c(v, u) = 0$.

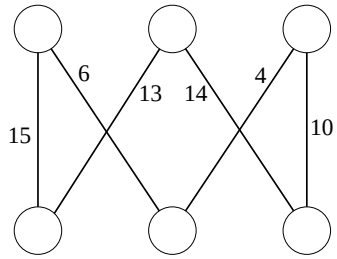
It is easy to see that with these modifications a graph G' is constructed which can be used to find the next augmenting path with minimal weight by using Dijkstra's algorithm. This process (the computation of shortest paths in G' including a shortest path p from s to t, inverting the augmenting path with minimal weight corresponding to p in G, modifying G' as mentioned above) is repeated until there are no more paths from s to t in G'. Then the current matching in G is the wanted matching of maximal cardinality with minimal weight among all those such matchings.

Remark: Because the presented algorithm works in the presence of negative edge weights as well, it is easy to find a matching of maximal cardinality with maximal (instead of minimal) weight among all those matchings as well. To this end, it suffices to just negate all edge weights before starting the algorithm.

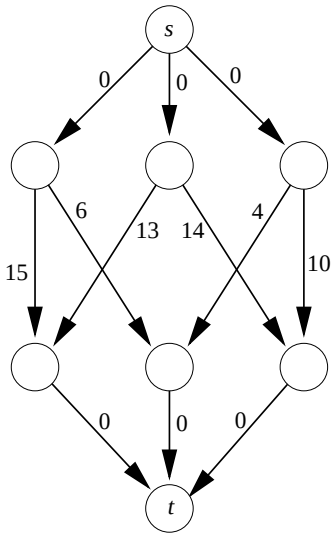
Further information: This design of the algorithm for finding matchings of maximal cardinality with respectively minimal or maximal weight is based on the pages 81–85 from the script "Advanced Algorithms" of a lecture by Johan Håstad. This script is available as a PostScript file on the web on Håstad's homepage `http://www.nada.kth.se/~johanh/`.

1.3 Example execution of the algorithm

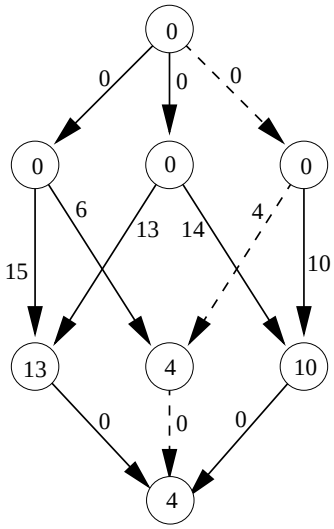
In the following we want to take a look at an example for the algorithm step by step.



(1) Input graph G

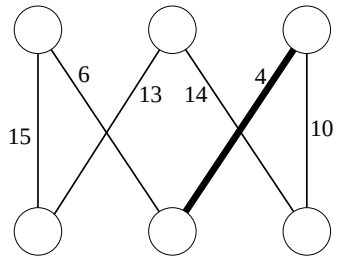


(2) Graph G'

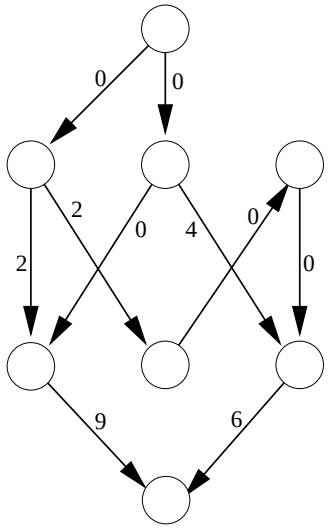


(3) Shortest paths in G'

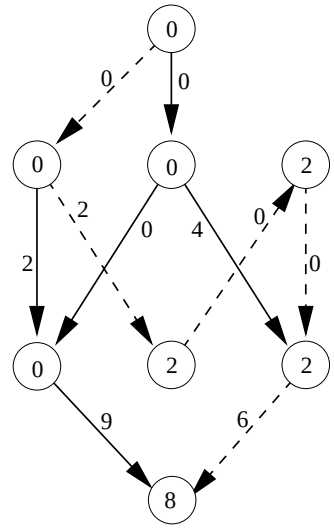
The input graph G is pictured in (1). The edge weights of G are all positive, such that no modifications of the weights is necessary beforehand. The graph G' is constructed as mentioned in section 1.2 and is pictured in (2). After the shortest paths from s to the other vertices in G' are computed using Dijkstra's algorithm, the distances of these vertices from s are pictured in (3), and the shortest path p from s to t is pictured by a dashed line. Inverting the corresponding augmenting path in G yields the matching M_1 pictured in (4), which has minimal weight among all matchings with exactly one edge.



(4) Matching M_1 in G



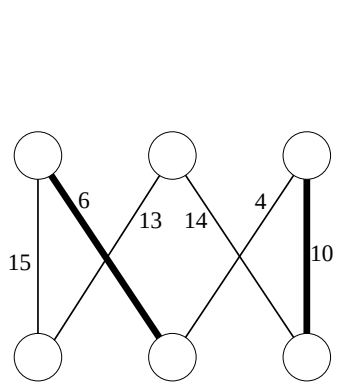
(5) New graph G'



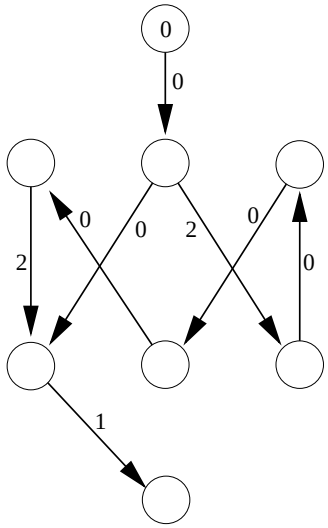
(6) Shortest paths in G'

By deleting the first and last edge of p, inverting the remaining edge of p and mo-

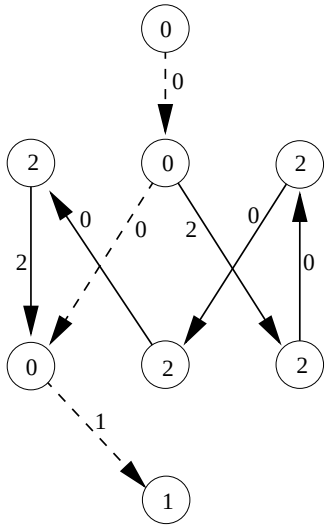
difying the weights, the old graph G' in (3) results in a new graph G' in (5). We execute Dijkstra's algorithm on this graph again; yielding the distances pictured in (6) as well as a new shortest path p (pictured by a dashed line) from s to t. Inverting the corresponding augmenting path in G yields in matching M_2 pictured in (7), which has minimal weight among all matchings with exactly two edges.



(7) Matching M_2 in G

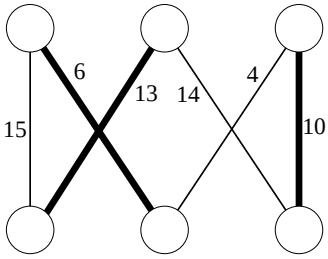


(8) New graph G'



(9) Shortest paths in G'

Again, the new graph G' is pictured in (8) and the result of the computation of the shortest paths in G' is pictured in (9). Inverting the augmenting path in G corresponding to the dashed shortest path G' finally yields the matching M_3 in (10). There are no more paths from s to t in the resulting graph G'. Hence, M_3 has maximal cardinality and minimal weight among all matchings of maximal cardinality in G, namely $w(M_3) = 29$.



(10) Matching M_3 in G