Minimum Cost Edge Disjoint Paths

Theodor Mader

15.4.2008

1 Introduction

Finding paths in networks and graphs constitutes an area of theoretical computer science which has been highly researched during the past decades. Depending on the types of networks and graphs, and the constraints on the path finding, the problem can vary a lot in terms of computational complexity. The task of finding the shortest path between two nodes in a directed graph, for instance, can be solved very efficiently, whereas the task of finding the **longest** path between two nodes turns out to be NP-complete.

In this document we describe an algorithm for computing minimum cost edge disjoint paths in a directed graph. The algorithm relies mainly on Menger's theorem, which establishes a connection of the disjoint paths problem to the minimum cardinality s-t cut problem. The minimum cardinality s-t cut of a graph is in turn equivalent to the maximum s-t flow, which can be computed in polynomial time. [3] provices a comprehensive overview of Menger's theorem and its connection to maximum flows.

2 Graph Definitions

Please note that in the following descriptions we focus mainly on graph properties special to our setting. We assume that the reader is familiar with the usual definitions of directed graphs, flow networks, flows and paths.

2.1 Directed Graphs

Let $G = (V, E, C)$ denote the graph consisting of vertices $V = \{v_1, ..., v_N\}$, and directed edges $E = \{(u, v) | u, v \in V, \text{ and } u \neq v\}$. With each edge $e = (u, v)$, we associate a cost $c(e)$. For the sake of simplicity we further assume that G does not contain opposite edges, i.e. if $(u, v) \in E$, then $(v, u) \notin E$ holds. This assumption simplifies the following constructions, but does not restrict the generality of the algorithm, as there exist procedures which convert graphs with opposite edges into graphs where the property holds. The vertex splitting procedure, for example, eliminates opposite edges, while at most doubling the number of vertices in the graph. Asymptotic complexity bounds thus still hold.

2.2 Flow Networks

We convert the graph G into the flow network $F = (V, E, C, 1, s, t)$, by copying the vertices, edges and edge cost of G, and additionally associating each edge $(u, v) \in E$ with the capacity 1. We also choose two designated vertices $s \in V$, and $t \in V$, $s \neq t$ as source and sink vertices. Note that this network has the special property that all edge capacities are set to 1.

2.3 Residual Networks

For a given flow network F, and a valid flow f on this network, we can construct the residual network $R_f = (V, E \cup E^{opp}, C \cup C^{opp}, s, t, r)$, by performing the following steps:

- Copy vertices, edges, edge cost, source, and sink from F
- For each edge $e = (u, v) \in F$, we add an opposite edge $e^{opp} = (v, u)$ to R_f, and assign e^{opp} with cost $c(e^{opp}) = -c(e)$
- We associate all original edges of R_f with the residual capacity $r(e) = 1 - f(e)$, all opposite edges with $r(e^{opp}) = -f(e)$.

2.4 Edge Disjoint Paths

We represent a path P_i with length $|P_i| = L$ in a given graph $G = (V, E, C)$ by the sequence of edges constituting the path, $P_i = \{e_{i1}, ..., e_{iL}\}$. A set of paths $\mathcal{P} = \{P_1, ..., P_K\}$ is said to be edge disjoint, iff it holds that

$$P_i \cap P_j = \emptyset \qquad \text{for all distinct pairs of paths } (P_i, P_j) \in \mathcal{P}.$$

Or, more informally speaking, a set of paths is edge disjoint iff each edge in the graph belongs to at most one path.

2.5 Minimum Cost Paths

For a set of paths $\mathcal{P}$, we compute the total cost $C(\mathcal{P})$, by summing up the cost of all edges in $\{P_1, ..., P_K\}$,

$$C(\mathcal{P}) = \sum_{i=1}^{K} \sum_{j=1}^{|P_i|} c(e_{ij}).$$

We say $\mathcal{P}$ is of minimum cost, if $C(\mathcal{P})$ is smaller (or equal) than the cost of all other sets of edge disjoint paths that contain the same number of paths as $\mathcal{P}$.

3 Algorithm

Having defined our notions of graphs and paths, we now derive the algorithm for computing the minimum cost edge disjoint paths from basic theorems and algorithms in graph theory. In particular, we use Menger's theorem to convert the disjoint paths problem into a minimum cut problem. The famous max-flow min-cut theorem is then used to further rewrite the problem as a maximum flow problem. A modified version of the Ford-Fulkerson [2] algorithm is finally used to solve the problem in polynomial time, under consideration of path cost.

3.1 Relating Disjoint Paths to Maximum Flows

Menger's theorem, rewritten for undirected graphs (see [3]) states:

Theorem 1 *Let $G = (V, E)$ be an undirected graph and let $P, Q \subseteq V$. Then the maximum number of pairwise disjoint $P - Q$ paths is equal to the minimum cardinality n of any set of vertices that intersects each $P - Q$ path.*

It provides thus an equivalence between the maximal number of $P-Q$ paths, and the minimum cardinality of an $P-Q$ cut. Note that the theorem can be proven for both, vertex and edge cuts.

In 1954, Ford and Fulkerson publish the famous Max-Flow Min-Cut theorem [2] for flow networks with source s and sink t.

Theorem 2 *The maximal value of the $s-t$ flows is equal to the minimal capacity of the $s-t$ cuts.*

This theorem directly relates the value of a maximal flow in the network to the minimal capacity of an arbitrary $s-t$ cut.

By associating each edge in the graph G of theorem 1 with capacity 1, the capacity of any $s-t$ cut becomes equivalent to the number of edges included in the cut, i.e. the cardinality of the cut. Both theorems are thus applicable and we can conclude corrolary 1:

Corollary 1 *The maximal $s-t$ flow in a network where all edge capacities are set to* 1 *is equivalent to the maximal number of edge disjoint $s-t$ paths.*

By the Ford-Fulkerson algorithm we know that as long as there exists a directed $s-t$ path in the residual network R_f of a given network F, the current flow f is not maximal. It can be increased by sending the maximal amount of flow possible along that path. For our case, all edge capacities are set to the value 1, hence, each step of the Ford-Fulkerson algorithm can at most send a flow of 1 along the augmenting path. Since this procedure directly saturates each edge on the path, we obtain observation 1:

Observation 1 *Each edge can be part of at most one directed path from s to t.*

Combining corollary 1 and observation 1 thus directly results in a simple algorithm for computing the maximal number of edge disjoint $s-t$ paths:

Algorithm 1 Compute maximal number of edge disjoint $s-t$ paths

$F \leftarrow$ convert input graph G to a flow network, as shown in 2.2
$f \leftarrow$ compute the maximal $s-t$ flow by the Ford Fulkerson algorithm
return all edges $\in F$ with flow 1

3.2 Minimum Cost Disjoint Paths

We now show how the proposed algorithm can be extended to take edge cost into account, thus yielding the maximal number of edge disjoint $s-t$ paths with total minimal cost. Cost computation and minimal cost paths are defined in section 2.5. The proposed algorithm builds upon two theorems proven in [1], pages 135-136.

Theorem 3 *Let f be a feasible flow in the network $F = (V, E, C, 1, s, t)$. Then f is a minimum cost flow iff the residual network R_f contains no cycles of negative cost.*

Thus, in order to be cost-optimal, we have to guarantee that our computed flows are cycle free. Theorem 3.10.5 in [1] (rewritten for the current setting, and to emphasize clarity) additionally states:

Theorem 4 *Augmenting the flow along the minimum cost $s-t$ path in the residual network R_f is optimal, i.e. no negative cost cycles are created.*

This lets us create a modified version of the Ford Fulkerson algorithm, which maximizes the flow and at the same time minimizes the total cost: We simply have to select the augmenting paths of the Ford Fulkerson algorithm according to minimal cost. Since our initial network is cycle free, and augmenting along the minimum cost path does not create cycles, we are guaranteed that the residual network remains cycle free during the whole execution of the algorithm. Combined with the fact that pushing flow along directed $s-t$ paths in R_f increases the total flow, we obtain a simple algorithm for computing minimum cost disjoint paths between s and t.

Algorithm 2 Compute minimum cost edge disjoint $s-t$ paths

```
%Prepare graphs
F ← convert input graph G to flow network, as shown in 2.2
R_f ← convert F to the residual network, assuming 0 flow.

% Ford Fulkerson main loop
while there exists a directed s − t path in R_f do
   P ← minimum cost s − t path in R_f
   f ← augment flow by 1 along P
   R_f ← compute residual network for new flow f
end while

return all edges ∈ F with flow 1
```

4 Complexity

Let $n = |V|$ and $m = |E|$. There can exist at most $|V|$ edge disjoint $s - t$ paths in the graph, hence the maximal possible flow is less or equal to n. Assuming that each augmentation along a directed $s - t$ path increases the flow by 1, the algorithm performs $O(n)$ path searches and augmentations. In full generality we have to assume that the edge costs may be negative, and thus we have to use the Bellman-Ford shortest path algorithm to perform each path search. The Bellman-Ford shortest path algorithm takes $O(mn)$ time, and we obtain a total worst case running time of $O(n^2m)$. Note that for the case where all costs are fixed to 1, the edge costs can be made positive, and Dijkstra algorithm can be used. With a running time of $O(n \log m)$, worst case complexity of our algorithm becomes $O(n^2 \log m)$.

References

[1] Jrgen Bang-Jensen and Gregory Gutin. *Digraphs - Theory and Applications*. Springer-Verlag, London, 2007.

[2] Jr. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[3] Alexander Schrijver. Paths and flows - a historical survey. 1994.