lem-in

Alexander Mayorov

I Introduction

This overview aims to document my work on a student project at School 42. For a comprehensive description of the problem, please consult the subject.

In short, the goal of the project is to move a given number of ants from the `start` node to the `end` node in a node-disjoint undirected network, where all nodes and edges have unit capacity, using the least number of steps. A step is a unit of time, during which each ant can move from the node it is currently in to an adjacent node, if that is available. Many ants can move simultaneously, but no collisions between them should ever occur.

This document is structured as follows. I first search for an effective solution to a simpler problem of edge-disjoint minimum cost flow in Sections II and III. I then adapt the algorithm in Section IV for the problem of node-disjoint minimum cost flow, which is more closely related to my actual problem. I then put everything together in Sections V and VI.

II Edge-disjoint minimum cost flow (basic)

Let's start with the problem of edge-disjoint minimum cost flow.

Definition 1. An active path is a unbroken sequence of edges, along which ants move at a given value of the flow f.

Definition 2. Flow constraints stipulate that at every node other than `start` and `end`, the amount of incoming flow equals the amount of outgoing flow.

Note that in our particular case, where all edges and nodes have unit capacities, any path from `start` to `end` or a closed path, such as a cycle, automatically satisfies the flow constraints.

Definition 3. A residual network G_r is constructed from a network G by reversing all edges along active paths and negating their cost. [2]

Observation 2.1. A cycle in the residual graph G_r doesn't change the value of the flow in the graph G, but can decrease / increase the cost of the flow by the amount equal to its own cost. [3]

Observation 2.2. A given flow f in the network G has minimum cost if there exist no negative cost cycles in the corresponding residual network G_r. [3]

Observation 2.3. The shortest path from `start` to `end` in the graph G doesn't introduce any negative cycles to the corresponding residual graph G_r.

This observation is true, because in the residual graph the negative cost of going from `end` to `start` along the shortest path is less than or equal to the positive cost of going from `start` to `end` along any of the remaining paths, by virtue of them not being the shortest.

Observation 2.4. A minimum cost flow of a given value (less than the overall max-flow of the network) can be obtained iteratively, by searching for the shortest path and replacing the network with its residual. Each iteration increases the value of the flow by 1.

Observation 2.5. The time complexity of getting a minimum cost flow of value f is $\mathcal{O}(fVE)$, where V is the number of vertices in the graph and E is the number of edges.

To convince yourself that this is true, observe that the presence of negative cost edges in the residual network precludes the use of the more efficient shortest path search algorithms, such Dijkstra's algorithm. Therefore, we must resort to running Bellman-Ford algorithm, whose time complexity is $\mathcal{O}(VE)$, at each iteration, of which there are f.

III Edge-disjoint minimum cost flow (efficient)

The time complexity of the iterative algorithm from Section II, $\mathcal{O}(fVE)$, becomes prohibitive when then the number of nodes gets sufficiently large (≈ 1000). Let's see if we can do better.

Definition 4. A price function is a function that assigns a numerical value to every node.

We can use a price function, in order to replace the original network G with a corresponding network G', in which the *effective* cost of an edge $c^*_{v\to w}$ is equal to

$$c^*_{v\to w} = c_{v\to w} + p(w) - p(v)\,,$$

where $c_{v\to w}$ is the original cost and $p(x)$ is the price of the node x.

Observation 3.1. Introducing a price function doesn't affect the presence or absence of negative cost cycles in the network, since the price terms around the cycle cancel out. [3]

Observation 3.2. If we use the distance to each node from `start` as its price, we can get rid of negative cost terms in the residual network. [3, 4]

To see that this is correct, note that, along the shortest path in the graph, the effective cost of an edge $v \to w$ is 0. Since we're using the shortest path to augment the flow, the reversed edges in the residual graph also have an effective cost of 0.

Observation 3.3. Searching for a minimum cost flow of a value f using distance as the price function has the time complexity $\mathcal{O}(fE \log V)$, since, in the absence of negative cost edges, we can use Dijkstra's algorithm to get the shortest path.

IV Node-disjoint minimum cost flow

The latter variant of the edge-disjoint min-cost flow algorithm can be modified using a clever trick to get paths that are not only edge-disjoint, but also node-disjoint.

Observation 4.1. We can split each node along the active paths, with the exception of the `start` and `end` nodes, into (1) an *IN* node, to which edges point to, and (2) an *OUT* node, from which edges originate. If we do that, we can enforce that as soon as the search branches onto one of these nodes, the only way out is via one of the inverse edges in the residual graph. [1, 4]

Please note that, as well as "splitting" nodes that lie on active paths, we should "unsplit" a node, if, in the course of the algorithm, there is not a single active path left that passes through it.

V Putting it all together

The lem-in problem can be reduced to the problem of node-disjoint minimum cost flow. However, the two problems are not exactly equivalent.

First, the number of ants is fixed. Therefore, the maximum flow in the graph may not give us the best possible solution. For example, in the case of a single ant, a single path of length 5 ($f = 1$) is better than a hundred paths ($f = 100$), of which the shortest one is 80 rooms long. Therefore, we must consider all possible values that the flow can take less than or equal to the number of ants in order to guarantee the optimal solution.

Can this be optimized?

Second, once we have the active paths, we need to optimally assign our ants to them, so as to minimize the number of steps from `start` to `end`. How we go about it is discussed in Section VI.

VI Assigning ants to paths

Observation 6.1. If there are n ants travelling along a single active path of length a, it would take them $(a + n - 1)$ steps to reach the `end`.

Because all nodes have a capacity of 1, after k steps at most k ants could have left the `start` and entered the path. The first ant reaches the `end` at the ath step, whereas the last one does so at the $(a + n - 1)$th step.

Observation 6.2. If we have two active paths of lengths a and b, respectively, and n ants, then, in order for all ants to get to the `end` as quickly as possibly, we should assign $(a - b)$ more ants to the first path than to the second.

Therefore, $\frac{n+(a-b)}{2}$ ants should walk the first path and at most $\frac{n-(a-b)}{2}$ should walk the second. It also follows that it will take at most $(a + \frac{n+(a-b)}{2} - 1)$ to reach the `end`.

The above observation holds in general, for any number of active paths. If we have M paths, of which the longest has length z, and n ants, then the jth path of length a_j should be walked by

$$n_j = z - a_j + \left[n - \sum_{i=1}^{M}(z - a_i)\right]/M\,,$$

ants, and the number of steps that the solution would take is equal to

$$\begin{aligned} N_{\text{steps}} &= a_j + z - a_j + \left[n - \sum_{i=1}^{M}(z - a_i)\right]/M - 1 \\ &= z - 1 + \left[n - \sum_{i=1}^{M}(z - a_i)\right]/M\,. \end{aligned}$$

Observation 6.3. If we obtain new paths in order of increasing length, we can stop considering any more paths, once the length of the next path is greater than the number of steps in the solution that uses only the paths considered thus far.

Note that this observation does **not** suggest that we can stop increasing flow in the network as soon as the above condition is satisfied, because the paths may change as the flow changes. All possible flow values must be considered, as it is stated in Section V. It only means that, for a flow of a **given** value, we can shortcut our calculation of the optimal assignment of ants among the active paths.

References

[1] http://www.macfreek.nl/memory/Disjoint_Path_Finding

[2] https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/

[3] https://courses.csail.mit.edu/6.854/06/scribe/s12-minCostFlowAlg.pdf

[4] https://en.wikipedia.org/wiki/Suurballe%27s_algorithm