LP. Lecture 9: Chapter 13: Network flow problems, continued.2

Let's do the last part about (MCF): the minimum cost network flow problem.

- Will sum up the algorithm.
- Look at a few details.
- Some combinatorial applications.

The primal network simplex algorithm

Summary of the algorithm: We start with a spanning tree T where the corresponding tree solution x_B is feasible, which means that $x_B \geq O$.

- Check optimality by calculating the dual variables y and z and check if $z \geq O$.
- If not optimal: do a pivot. This means choosing an edge (u, v) where $z_{uv} < 0$, and finding a new spanning tree T' by adding e and removing a certain other edge (so that T' becomes a spanning tree). Update the tree solution x.

The calculation of both x and y is done through "leaf elimination" by using the triangularity like we have seen. The calculation of each z_{uv} is made directly from the corresponding equation in the dual problem:

$$z_{uv} = c_{uv} + y_u - y_v.$$

Efficient updating of the variables in a pivot

Updating of x: We don't need to start all over again when a new x is to be calculated. The reason is that x only changes for certain edges. The new edge e (the ingoing basic variable) belongs to a cycle C in the graph. The change of x becomes

- Let the ingoing variable be equal to some number ϵ (which is going to be determined).
- For each edge $f \in C$ with the same direction as e in the cycle ("forward edges"), x_f will be increased by ϵ. In the edges in C that have the opposite direction ("backward edges") the flow will be reduced by ϵ. The flow in edges outside the cycle C remains the same.
- ϵ is chosen as the minimum of the flow in the backward edges.

Updating the spanning tree T: The ingoing edge e replaces a backward edge f in the cycle which has now gotten the value $x_f = 0$. This gives us a new spanning tree T'.

Comments:

1. There may be several backward edges that obtain the flow value 0; we then have a degenerate solution after the pivot. If one does not use a specific pivoting rule, one of these edges is simply chosen randomly as the outgoing variable.
2. Degeneration often occurs in (MCF) problems. Some claim that about 70-80% of the pivots tend to be degenerate! But, luckily, cycling is not considered a practical problem. Further, there is a pivoting rule (via so called strongly allowed spanning trees) that avoids cycling (and that in practice often reduces the number of pivots).

3. If the cycle C does not have any backward edges the problem is unbounded. So: we can by sending a suitable flow around in this cycle get the total cost to approach $-\infty$. (This usually does not happen i practical problems.)
4. As mentioned one often has upper bounds on x in (MCF) problems:

   $$0 \leq x_{uv} \leq a_{uv} \quad ((u, v) \in E).$$

   The algorithm can easily be adapted to this more general situation. Briefly, the changes are the following: Each x_e where $e \notin T$ (non basic variables) either has the value 0 or the upper bound a_e. When determining the maximal ϵ we also have to make sure that no forward edges receive a larger flow than their capacity. The nonbasic variables that are on their upper bound, can only be reduced, so the test of optimality must be extended in accordance to this. This means that such a variable can be taken into basis if the corresponding z-component is positive (while for nonbasic variables on their lower bound 0 they are candidates when z is negative there).

Updating of y: Assume that we are going to add e and remove f from our spanning tree T. $T \setminus \{f\}$ consists of two (sub)trees T_r and T' where T_r contains the root node r. Then the following applies for the update of the dual variables y:

- If the new edge e goes from T_r to T', all the dual variables in T' are increased by the same size Δ (equal to the value of the dual slack variable z_e, so $\delta < 0$). The variables in T_r remain unchanged.
- If the new edge e goes from T' to T_r, all the dual variables in T' are reduced by Δ. The variables in T_r remain unchanged.

Updating of z: The following applies:

- There is no alteration of z_{uv} if u and v lie in the same of the two subtrees T_r and T' (because y_u and y_v are altered by the same size).
- For edges between the two subtrees and that have the same direction as the ingoing edge e, z will be reduced by (the original) value z_e, while those that go in the opposite direction will be increased by z_e.

How to find an initial feasible solution?

So far, we have just assumed that we have an initial feasible solution, which means that we have a spanning tree so that the corresponding tree solution is nonnegative. Sometimes, one is lucky and observes such a solution directly, but usually it takes more work.

- There are several techniques for finding a initial feasible solution. One possibility is to use the dual simplex algorithm, adapted to the structure in the (MCF). We will explain this method next.
- We remark that another possibility is to reformulate the problem as a maximum flow problem: this is a basic network problem for which very fast algorithms have been developed.

The dual simplex algorithm for finding an initial feasible solution:

Step 1. Look at the (MCF) problem with a modified cost function, namely where $c = O$. Then, every spanning tree is dual feasible, so we choose a (random) spanning tree T. The basis is dual feasible, and if we're lucky it will also be primal feasible. In such a case, the job is done and we simply use T as the initial solution for our (MCF) algorithm with the original cost function c. On the other hand, if x has negative components, we go to step 2.

Step 2. Choose an edge $e = (u, v)$ with $x_e < 0$. We now make a pivot where e leaves basis. The reason is that we then have flow 0 in e (since the variable becomes nonbasic). The question is: which variable should go into basis. Look at the two subtrees T_r and T' which occur if e is removed from T (T_r contains the root node r). To get another spanning tree we have to choose an edge f that connects the two trees (have an end node in each tree). It is clear that f must go the in the opposite direction of e (otherwise, an increase of x_f would result in a lower x_e and we want to increase x_e). But there may be several such edges!

Step 2. (cont.) We then choose f with (the dual slack) z_f as small as possible among the edges between the two subtrees and with the opposite direction of e. The reason for this choice is that after the pivot we will have a new solution that is also dual feasible. So, all the candidate edges have their z-values reduced with the same size; we saw this earlier in connection to the update of the z-variable in a pivot. In this way, the outgoing edge e and the ingoing edge f is determined and we implement the pivot and update the variables as in the usual (MCF) algorithm. We repeat these pivots until x is nonnegative. (This is possible if the problem really has an feasible flow.)

We now leave algorithms and take a look at an interesting application of the theory in the area of combinatorics.

Application: combinatorics

We call a vector integral if all the components are integers. Here is the result:

Theorem *Consider the (MCF) problem*

$$\min\{c^T x : Ax = -b, \;\; x \geq O\}$$

where b is an integer, and assume that there is an optimal solution. Then, there exists an optimal solution x which is an integer.

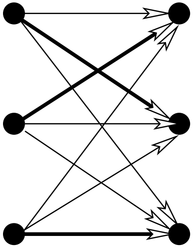
Proof: In each iteration the tree solution x is computed by leaf elimination. The first variable computed will then equal $\pm b_v$ which is an integer. Each of the next variables become a sum of certain integers (a certain b_v and $\pm$ other already computed flows that are integral). Therefore x is integral. $\square$

Comments:

- As explained in the proof all the flows x computed during the algorithm are integral, not just the optimal solution.
- If c is integral, there is an optimal dual solution (y, z) which is integral. (This can be sees directly from the algorithms for computing y and afterwards z.)
- The theorem above is also valid when we have integral capacities on x, so $0 \leq x_e \leq a_e$ $(e \in E)$ where each a is integral. So, in this situation there is also an integral optimal solution x.

This theorem has many applications in combinatorics (discrete mathematics). Let us have a look at such an application.

Consider the digraph



Let $b_v = 1$ for each of the three nodes to the left and $b_v = -1$ for the three others. Furthermore, we have certain costs for the edges, but we don't have to worry about those now.

- Since b is integral, there is an optimal flow x which is also integral (and we know how to find it).
- But based on the values in b we see that x must have exactly three components that equal 1, while the rest is 0.
- Further, the three edges where $x_e = 1$ are disjoint, which means that they have no endnodes in common. Such an set of edges (in an undireced graph) is called a perfect matching, see the edges with boldface lines in the figure.
- In this example there are $3! = 6$ different perfect matchings and they all correspond to allowed solutions in the (MCS) problem.
- But then it follows that the solution we found actually is a optimal perfect matching, ie. a perfect matching with minimum total cost.

- This problem is, naturally, called the minimum cost (weight) perfect matching problem in a bipartite graph (which is the graph we get if we ignore the direction of the edges; it is bipartite because each edge has an end node in each of the two nodesets (left and right)).
- In general we have n nodes to the left and n nodes to the right. The number of perfect matchings is then $n!$, which even for a moderate n is a gigantic number. So the method finds an optimal perfect matching in a problem where direct enumeration of all the possibilities is impossible. This is important!
- A practical application of perfect matching is when assigning jobs to people, computers or something else. Assume that each person can do exactly one job and that each job should be done by one person. The problem is then to find such an allocation of jobs with the lowest possible cost: this is called the assignment problem. There are also other variants of the simplex algorithm (also theoretically efficient) for this problem.

Other special cases of (MCF) that are important in many connections are

- the shortest path problem.
- the maximum flow problem.
- the transportation problem. This generalizes the bipartite perfect matching by general supply/demand b which is negative "to the left" and positive "to the right"; this may be given a transportation interpretation (find min. cost transportation plan from supplies to customers)! And this is a basic problem within application in *transportation optimization*.

Again, there are efficient combinatorial algorithms; more about these things in the fall course INF-MAT 5360 Mathematical optimization. There, you will also learn more about the mathematics that constitute the basis for analyzing/understanding/solving these problems.

You are invited there!