Appplications of Network Flow

Here are two sample problems which illustrate applications of network flow algorithms.

1. Given a computer network of n processors $P_1, P_2, \ldots, P_n$, and m communication lines $C_1, C_2, \ldots, C_n$, where the i-th processor can test t_i lines per day, we are asked to find a testing schedule or determine that no schedule can test all lines in a single day. We are given a set of tuples (P_i, C_j) indicating which processors can test which lines.

   Solution:

   Convert this computer network to a network flow graph by creating a source vertex s and adding edges (s, P_i) of weight t_i. Next add edges of unit weight (P_i, C_k), if the i-th processor can test the k-th communication line. Last, add a sink t and edges unit weight (C_k, t) for each of the communication lines. In order to find a testing schedule, simply run a max flow algorithm, and then BFS from t in the residual graph and see how many communication lines are reachable from t. If all are reachable, then see which processor is testing which line and that is the schedule. If not all lines are reachable, then no schedule exists.

   In terms of the running time analysis, the best max flow algorithm available to us is CLRS section 26.5, the relabel-to-front max flow algorithm, which runs in $O(V^3)$. Thus, our algorithm for determining whether all lines can be tested in a day is $O(V^3)$. Note: we assume all communications lines are attached to at least one processor.

2. Now we are asked to find the minimum number of days needed to test all of the lines, without finding a schedule.

   Solution:

   We augment the algorithm in the previous problem in the following manner:

   1. Let $k = 1$.
   2. Construct the graph as specified above with edge (s, P_i) having capacity $k \cdot t_i$.
   3. Run the max flow algorithm. Upon termination, run a BFS from t to see if all lines are reachable from t.
      a. If all lines are reachable, find which processors can test which lines and return k.
      4. If not all lines are reachable, then let $k = k + 1$ and repeat from step 2.

   When our algorithm returns, we only know that P_i can test a given set of lines, but we don't know in which order. However, the order is irrelevant. As long as P_i tests only those particular lines, t_i per day, than the solution will be optimal. This algorithm is $O(mV^3)$, because it would take at most m days to test all m lines. However, we can improve this by doing a binary search and taking k as $\{1, 2, 4, \ldots\}$. Thus, we can improve the time to $O(\log dV^3)$, where d represents the number of days needed to test all lines. Again, we are using the relabel-to-front $O(V^3)$ algorithm as our underlying means for determining max flow.