A Polynomial-Time Algorithm for Bandwidth-Optimization

Josh Frankel, Matt Nettleton, and Steve Marx

March 8, 2002

Abstract

The bandwidth-optimization problem is a real-world instance of the job/station problem. In the job/station problem, we're given a set U and N subsets of U. Two of these subsets are called stations, and the rest are called jobs. When a job is assigned to a station, the union of the two subsets becomes the new content of the station. Each job is assigned to exactly one station. The problem is to find a set of job assignments such that the sum of the number of elements in the two stations is minimal.

In this document, we present a polynomial-time algorithm to solve this optimization problem by casting the problem as an instance of the digraph minimum cut problem.

1 Introduction

Casting the job/station problem in the minimum cut domain can be viewed as a two stage process. First, we can reduce the problem to that of a minimum vertex-cut in an undirected graph. Here we define the minimum vertex-cut of an undirected graph as finding a minimal subset of the vertices (not including two vertices designated the source and sink) such that when these vertices (and all incidental edges) are removed from the graph, no path exists from the source to the sink.

Second, we can show an equivalence between the minimum vertex-cut problem in an undirected graph and the minimum cut in a related directed graph.

2 Job/station as Minimum Vertex-Cut

First we recognize that the job/station problem can be viewed as minimizing the overlap between the two stations. In the best case, we have $|S_1 \cup S_2| = |U|$ elements total in the two stations where $S_i \subseteq U$ is the set of elements in station i. The sum of the sizes of the two stations is then $|U| + |S_1 \cap S_2|$. To minimize the total size of the two stations, we need to minimize their overlap $|S_1 \cap S_2|$.

Further, we recognize that for any element $l \in S_1 \cap S_2$, removing l from the problem does not change the solution. This is because for any job, its assignment to S_i will not add l to it, as both stations already contain l. Therefore, which station to assign a job to does not depend on its containing an element common to both stations.

An algorithm begins to present itself. We can keep removing elements from the problem until no job assignments cause $|S_i|$ to increase. The resulting $S_1 \cap S_2$ will consist of the elements L we removed from the problem. We thus need to minimize the number of elements we remove. We're now ready to cast the problem as a minimum vertex-cut problem. Algorithm 2.1 gives pseudo-code to solve the problem.

Algorithm 2.1 *Construct a graph G as follows:*

1. *For each element $l \in U$: G.addVertex(l)*

2. *For each station S_i: G.addVertex(S_i)*

3. *For each job $J_i \subseteq U$:*

   For each pair of elements $l1, l2 \in J_i$: G.addEdge($l1$, $l2$)

4. *For each station S_i:*

   For each element l in S_i: G.addEdge(S_i, l)

Find the minimum vertex-cut (S_1, S_2) of G, which corresponds to the optimal solution of the job/station problem.

Notice that instead of creating vertices for each J and edges from each J_i to the l's it contains, we've simply connected the l's directly. This simplification does not change the result; it only makes the resulting graph more manageable. (It is easy to see that the connectivity is not affected by this simplification, and we want to disallow removing jobs from the problem anyway.)

Now we need simply to solve the minimum vertex-cut problem, and we will have a complete solution to the job/station problem.

3 Minimum Vertex-Cut as Minimum Cut

The minimum vertex-cut problem is formally stated as follows:

Definition 3.1 *Given an undirected graph $G = (V, E)$ and two vertex designations s and t, find a minimal set $V' \subseteq \{V\} - \{s, t\}$ such that in $G' = (\{V\} - \{V'\}, \{E\} - \{e' \in E | \exists_{v'}$ e' is incidental to v' $\})$, no path exists from s to t.*

We can cast this problem relatively easily as a minimum cut problem in a directed graph. We simply create edges in either direction for each undirected edge in the original graph. Then we split each vertex l into two, l_in and l_out. All edges that were incoming to l are now incoming to l_in, and all edges that were outgoing from l are now outgoing from l_out. A single directed edge is added from l_in to l_out.

Now it is clear that cutting an edge between l_in and l_out in this graph is equivalent to removing the vertex l in the undirected graph, because now all paths that used l are disconnected.

This formulation gives us a simple algorithm to extend Algorithm 2.1 to provide a complete solution. Algorithm 3.1 shows pseudo-code for the complete problem.

Algorithm 3.1 *Construct a directed graph G as follows:*

1. *For each element $l \in U$: G.addVertex(l_{in}) and G.addVertex(l_{out})*

2. *For each station S_i: G.addVertex(S_i)*

3. *For each job $J_i \subseteq U$:*

   For each pair of elements $l1, l2 \in J_i$: G.addDirectedEdge($l1_{out}$, $l2_{in}$)

4. *For each element l in S_1: G.addDirectedEdge(S_1, l_{in})*

5. *For each element l in S_2: G.addDirectedEdge(l_{out}, S_2)*

Find the minimum cut (S_1, S_2) of G, which corresponds to the optimal solution to the job/station problem.

One further step is needed to prove that this always yields a correct solution. We need to show that such a minimum cut can be directly mapped back into a complete set of job assignments.

To prove this, we first constrain our minimum cut solution to contain only "mid-edges," the edges that connect the *in* and *out* vertices of a given element. This doesn't effect the optimality of our solution, because all paths from S_1 to S_2 contain at least one such mid-edge, so we can achieve the same effect of removing any edge by removing a mid-edge along the same path to S_2.

Constraining our solution in this manner, we now have a list of elements that will be in $S_1 \cap S_2$, the elements whose mid-edges have been cut. This is an unambiguous representation of a set of job assignments, because now we need only to assign each job J_i to S_1 or S_2, whichever already contains all the elements of J_i. If both stations satisfy this requirement, the choice can be made arbitrarily, as it doesn't effect $|S_1 \cap S_2|$, as discussed in Section 2.

4 Run-Time Analysis

To analyze the efficiency of the algorithm presented here, we first describe the overall algorithm as a set of discrete steps and then analyze the run-time of each step individually. The pseudo-code in Algorithm 4.1 gives the breakdown of steps.

Algorithm 4.1

- *Given U, $N-2$ jobs, and two stations, construct a graph G as given in Algorithm 3.1.*
- *Given this graph G, find the minimum cut.*
- *Given a minimum cut of G, output the job assignments for each job.*

We tackle each step independently.

4.1 Graph Construction

Graph construction consists of creating $|U|+2$ vertices, one for each element and one for each station. We then add edges for each pair of elements in each job. In the worst case, each job contains all the elements of U, so there are $|U|(|U|-1) = |U|^2 - |U|$ edges for each of the N jobs, or a total of $N(|U|^2 - |U|)$ edges. So the graph construction step has complexity $O(|U| + 2 + N(|U|^2 - |U|) = O(N|U|^2)$.

4.2 Minimum Cut

We can find the minimum cut by the Edmond-Karp method, using the difference between the original graph and the residual graph to find the minimum cut set. The time complexity of the Edmond-Karp algorithm is $O(VE^2)$ as given in [1]. For our purposes, we have $|U|+2$ vertices and $|U|^2 - |U|$ edges, so the time complexity of this step is $O((|U|+2)N(|U|^2 - |U|)^2 = O(N|U|^5)$.

4.3 Job assignments

We add all elements whose mid-edges were cut in the minimum cut to each station first. This is $O(|U|)$. We then loop over each job, checking that all of its elements are contained within the first station. Assuming a hash table or other constant-time look-up is used, we can check each job in time proportional to the size of the job. In the worst case, this size is simply $|U|$, so there are $N|U|$ total elements to check, an $O(N|U|)$ operation. Summing the times, we get a total of $O(N|U| + |U|) = O(N|U|)$ for this step.

The total time complexity is given by the sum of the complexities of each step. The sum is dominated by the minimum cut time, so the overall complexity is $O(N|U|^5)$.

5 Conclusion

The algorithm given here is a polynomial time solution to the job/station problem for two stations. For more stations, the problem can be cast as a multi-way cut problem, which in NP-hard for $k > 2$.

The solution presented in this paper is not necessarily optimal. We have no proof guaranteeing that no faster algorithm exists. Indeed, in the minimum cut step, we believe that because all edges are weighted equally, there may be a faster method. The purpose in formulating this algorithm was merely to show that a polynomial time algorithm exists and to give one possible implementation.

References

[1] Cormen, T., C. Leiserson, and R. Rivest. *Introduction to Algorithms.* The Massachusetts Institute of Technology, 1990.