Addressing the Problem of Finding a Single Vital Edge in a Maximum Flow Graph

Barton, Alan

*Addressing the Problem of Finding a Single Vital Edge in a Maximum Flow Graph**

Barton, A.
November 2005

* published as NRC/ERB-1129. 4 pages. NRC 48305.

National Research Council Canada
Institute for Information Technology

Conseil national de recherches Canada
Institut de technologie de l'information

ERB-1129

NRC·CNRC

Addressing the Problem of Finding a Single Vital Edge in a Maximum Flow Graph

Barton, A.
November 2005

Canada

NRC 48305

Addressing the Problem of Finding a Single Vital Edge in a Maximum Flow Graph[*]

Alan J. Barton[†]
Integrated Reasoning Group
Institute for Information Technology
National Research Council Canada
Ottawa, Canada, K1A 0R6
alan.barton@nrc-cnrc.gc.ca

ABSTRACT

A first attempt is made to understand some of the theoretical considerations in the design of a solution to the problem of finding a vital edge in a flow graph G. Examples are used to conjecture the existence of four equivalence classes for the set of all possible flow graphs; both of which (counter examples and partitions) are presented in sufficient detail as to allow the possibility of potentially deriving a specific algorithm and hence implementation.

Keywords

vital edge, maximal flow, algorithm design

1. INTRODUCTION

The maximal flow problem [3] was formulated by T. Harris as follows:

> Consider a rail network connecting two cities by way of a number of intermediate cities, where each link of the network has a number assigned to it representing its capacity. Assuming a steady state condition, find a maximal flow from one given city to the other.

Which states that a linear programming problem formulation could be made [3], and thus solved (not necessarily most efficiently in all cases) by the simplex method [2].

2. PROBLEM STATEMENT

This paper attempts to propose a solution to the problem as stated by Prof. Maheshwari: Assume that we have a network flow graph $G = (V, E)$ with positive capacities on each of the edges and two specified vertices s and t. Suggest an efficient algorithm to find an edge in E, such that setting its capacity to zero (i.e. deleting this edge) will result in the largest decrease in the maximum flow in the resulting graph.

2.1 Related Problems

It was found after the assignment submission, that *i)* the search for an optimal arc can be restricted to one of the $n - 1$ arcs in any maximal spanning tree[1] and *ii)* in the network inhibition problem[5], we wish to find the most cost-effective way to reduce the ability of a network to transport a commodity. Potential applications include disabling polluting pipe systems, military supply lines, and illicit drug networks. We wish to compute a fixed-budget attack strategy which will maximally inhibit a network's capability.

Further, a related problem to the one of *vital edge deletion*, is one of *vital node deletion*[4], which can be stated more formally [4] as follows: In an undirected, 2-node connected graph G=(V,E) with positive real edge lengths, the distance between any two nodes r and s is the length of a shortest path between r and s in G. The removal of a node and its incident edges from G may increase the distance from r to s. A most vital node of a given shortest path from r to s is a node (other than r and s) whose removal from G results in the largest increase of the distance from r to s.

3. CONSIDERATIONS

Four cases will be considered, from simple, to more complex, in the design of a solution for the stated problem. Counter examples will be used in an attempt to justify the consideration of more complex scenarios in the design. All cases and examples will be fully explicated to the best of the author's ability under the constraints imposed by the scope and duration of the course.

In all descriptions, there exist unique nodes s and t, called the source and sink respectively, where flow (measured as the quantity f_i) emanates from s and is attracted to t. The restriction is imposed that not too much flow can move at any given time along an edge e_i; which is called the capacity (c_i) of e_i.

[*]Problem from Assignment 2 for the course COMP5703 entitled *Design and Analysis of Algorithms* at Carleton University

[†]Master of Computer Science (in progress)

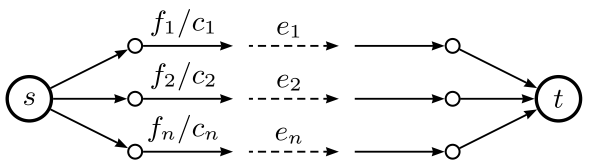

Figure 1: Case 1: Flow comes out of the source s and moves to the sink t. As soon as the flow splits to one of the incident edges, it never joins again until t is reached.

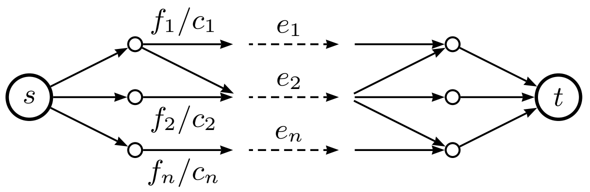

Figure 2: Case 2: Flow emanates from s and moves towards t, but may join or split with other flow zero or more times after leaving s.

3.1 Case I: No Cycles, Disjoint Paths

Case I is the simplest case in which all paths from s to t are disjoint, as can be seen in Fig-1. For the i-th path between s and t (call it P_i) the flow f_i will be equal to the $min_{e \in P}\{c_e\}$, where c_e represents the capacity of the edge e on the path P_i. Intuitively, this is due to the fact that no more flow than the smallest capacity edge along the path can be pushed from s to t.

This leads to the possibility that not all edges need to be considered when attempting to determine the vital edge e_v for deletion. Perhaps, only those incident to s are the only potential vital edges because if such an incident edge is deleted, then necessarily the flow must decrease by the capacity of that edge. This leads to the following algorithm:

```
FIND-A-VITAL-EDGE($G = (V, E)$ and $s, t \in V$)
1  Find the outgoing edge $e$ of $s$ with largest flow
2  Return $e$ as the vital edge $e_v$
```

3.2 Case II: No Cycles, Overlapping Paths

Case II relaxes the restriction that paths must be disjoint; Fig-2 provides a visual representation of this case.

When Fig-3 is used as input for FIND-A-VITAL-EDGE, it can be clearly seen to be incorrect. A modified version of the algorithm could be constructed in order to take this new example into account. The new version, FIND-A-VITAL-EDGE$^{(2)}$, is as follows:

```
FIND-A-VITAL-EDGE$^{(2)}$($G = (V, E)$ and $s, t \in V$)
1  Find the outbound edge $e_s$ of $s$ with largest flow
2  Find the inbound edge $e_t$ of $t$ with largest flow
3  Return the larger flow of $e_s$ and $e_t$ as the vital edge $e_v$
```

However, FIND-A-VITAL-EDGE$^{(2)}$ performs incorrectly when Fig-4 or Fig-5 is given as input. The observation is that if there exists a path from s to t that goes through one edge, then certainly that one edge would be a candidate for deletion. In the case of Fig-5, two such vital edges exist (e_1 and e_2) that would both reduce the maximum flow by two and that are both not incident to either s or t. The intuition is that the disjoint parts of the paths are being ignored with only the paths as a whole being considered, thereby, in some sense, reducing this example to one that could fit into the Case I analysis. Put another way, we need to somehow collapse Case II down to Case I via equivalence classes, and we can do this when they share an edge. This leads to FIND-A-VITAL-EDGE$^{(3)}$:

```
FIND-A-VITAL-EDGE$^{(3)}$($G = (V, E)$ and $s, t \in V$)
1  Find all paths from $s$ to $t$
2  Put all paths into the same equivalence class if they
   share edge $e_i$
3  Return the equivalence class with highest flow as
   the vital edge $e_v$ (break ties arbitrarily)
```

When FIND-A-VITAL-EDGE$^{(3)}$ is given Fig-6 as input, it would fail, because e_1 would be reported as the vital edge but e_1 or e_2 are certainly vital edges. What is different between e_1 and e_2 that makes deletion of e_1 have no effect on the maximum flow of the graph G? The capacity of e_1 is 3, which is the same as the in-degree of the node at the start of the edge e_1 (i.e. a difference of 0). The capacity of e_2 is 4, which is the same as the in-degree of the node at the start of the edge e_2 (i.e. again, a difference of 0). But we notice that if e_1 is deleted, then all of the flow can be re-routed to e_2, and have no net change on the maximum

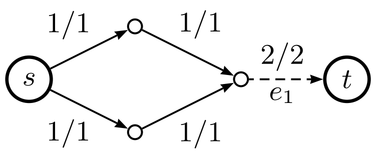

Figure 3: Counter Example: FIND-A-VITAL-EDGE cannot find the vital edge e_1 because it is not incident to s and participates in two paths simultaneously, breaking the assumption of Case I.

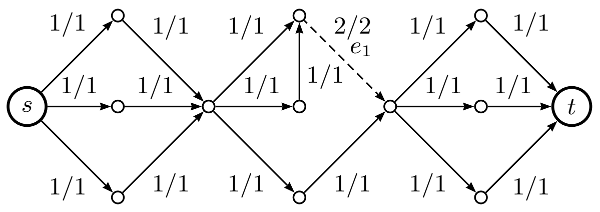

Figure 4: Counter Example: FIND-A-VITAL-EDGE$^{(2)}$ cannot find the vital edge e_1 because it is neither incident to s nor t. In this case e_1 has the highest capacity of all edges, but it need not. For example, change the capacity of one of the incident edges to s to 3 and the flow of G will be the same, but e_1 will now not have maximum capacity over G.

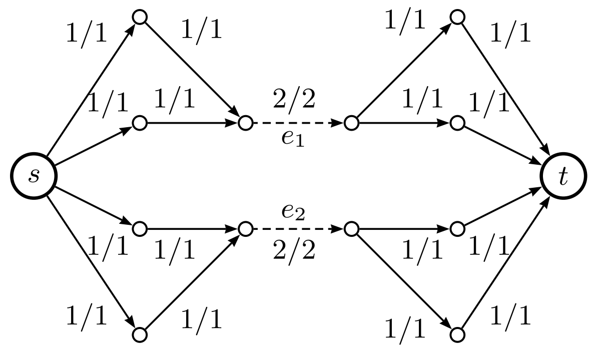

Figure 5: Counter Example: Find-A-Vital-Edge$^{(2)}$ **cannot find one of the vital edges e_1 or e_2 because both are not incident to either s or t.**

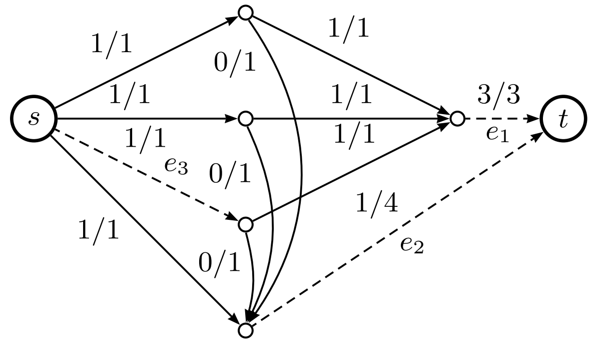

Figure 6: Counter Example: deletion of e_1 would not disrupt the maximum flow, whereas deletion of $e2$ or $e3$ would certainly decrease the flow by 1. In this case, the vital edge is not the highest flow edge, but is the highest capacity edge.

flow of G. How do we know that the edges e_1 and e_2 have such a re-routing property? Certainly our initial, somewhat naïve algorithm Find-A-Vital-Edge would be guaranteed to reduce the flow in these cases, but the reduction in flow would not necessarily be the maximum achievable.

And what about if the paths shared a vertex, rather than an edge? Fig-7 shows that the vital edges in G are dependant on the flow and the capacity, because changing the capacity values for the edges, changes whether the edges are vital or not.

One conjectured way to simplify G, when multiple edges enter and leave a vertex, would be to split off a new vertex v, and attach a subset of the incoming and outgoing edges to v as is illustrated in Fig-8. This new artificially constructed vertex, may (hypothesis) help simplify the analysis of the resultant graph G'. More specifically when referring to Fig-8, if $f_i = f'_j$ then we can construct a new node v to form a new, simpler graph G' when attempting to find a vital edge e_v. In general, we may need l in-edges with flow f_i^p, for $p \in [1..l]$ to match (i.e. equal sum flow totals) with k

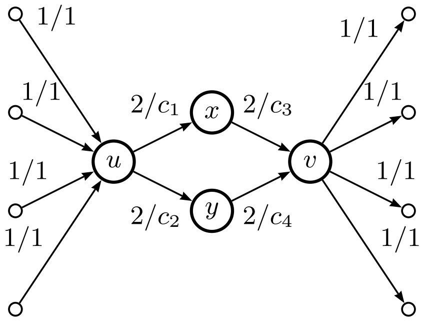

Figure 7: Observation: The edges ux, uy, xv and yv may or may not be vital edges, because they depend on the capacity values c_1, c_2, c_3 and c_4. For example, if $c_1 = c_2 = c_3 = c_4 = x$ and $x = 2$ then the edges will be vital, but if $x = 4$ then no one edge is vital.

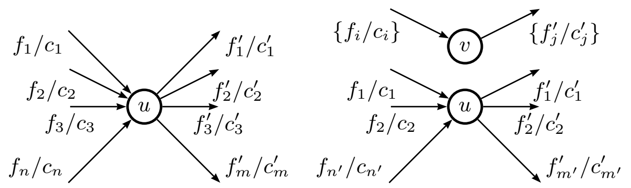

Figure 8: Idea: Construct a new node in order to simplify the paths.

out-edges with flow $f_j'^q$, for $q \in [1..k]$.

A specific example of simplifying G is given in Fig-9. The idea is to collapse paths $p_1 = \{e_1, e_3\}$ and $p_2 = \{e_2, e_4\}$ between nodes u and v because they all have the same flow and so none of the edges will be a unique vital edge. That is, collapse paths if and only if the max flow along each path between two vertices is equal. This could also be stated as collapsing triangles composed of k vertices.

3.3 Case III: Cycles, Disjoint Paths

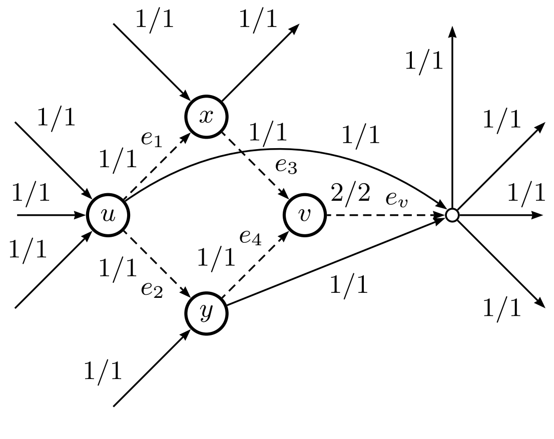

Figure 9: Simplification Conjecture: Collapse u v into one node when two distinct paths exist between u and v that all have the same flow.

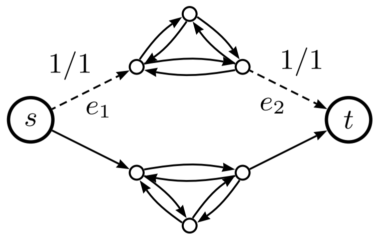

Figure 10: Case 3: The addition of cycles onto disjoint paths.

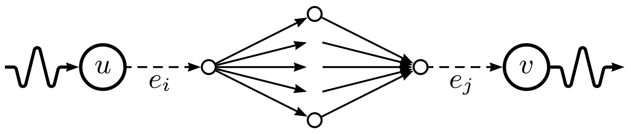

Figure 11: Idea: Entering (e_i) and exiting (e_j) edges are vital edge candidates.

The addition of cycles on a path from s to t means that there is still exactly one edge entering the cycle and one edge leaving the cycle, as in, for example, Fig-10, where e_1 enters and e_2 exits the cycle. This is a generalization from that of entering (exiting) a set of nodes through one edge, as is depicted in Fig-11, which was the key observation for Case I.

3.4 Case IV: Cycles, Overlapping Paths

The most complex flow graph has vertex and edge sharing on overlapping paths containing cycles. The cumulative descriptions given, may help derive an algorithm for Case IV, as depicted, simplistically, in Fig-12.

4. CONCLUSIONS

The simplification of the problem of determining the most vital edge in an input graph G through the construction of equivalence classes (partitions) on the set of all possible input graphs has been proposed. The specific graph G may be transformed through simplification transformations in order to determine its equivalence class. Such simplifications may aid the more efficient determination (rather than the naïve brute force approach) of a vital (not necessarily unique) edge of G. Detailed analysis and practical implementation, including specifying algorithms for cycle and path overlap detection, are left for future work. A generalization of the presented problem (suggested briefly by Prof. Maheshwari) would be to find k vital edges that maximally reduce the maximum flow of the graph G.

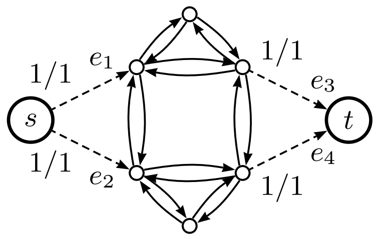

Figure 12: Case 4: e_1, e_2, e_3, and e_4 are the only edges that would definitely cause flow to decrease. This example could also be viewed as having two edges incident to a cycle, both entering and leaving.

5. ACKNOWLEDGMENTS

The author would like to thank *i)* Prof. Maheshwari for teaching the course, *ii)* NRC-IIT-IR group for supporting attendance at this course, in particular, the following people from NRC: Julio J. Valdés, Bob Orchard and Fazel Famili, and *iii)* CISTI for supplying research materials.

6. REFERENCES

[1] Y. P. Aneja, R. Chandrasekaran, and K. P. K. Nair. Maximizing residual flow under an arc destruction. *Networks*, 38(4):194–198, November 2001.

[2] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities: Activity analysis of production and allocation. 1951. Cowles Commission.

[3] L. Ford and D. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3):399–404, 1956.

[4] E. Nardelli, G. Proietti, and P. Widmayer. Finding the most vital node of a shortest path. *Theoretical Computer Science*, 296(1):167–177, 2003.

[5] C. A. Phillips. The network inhibition problem. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, number ISBN:0-89791-591-7, pages 776–785, New York, NY, USA, 1993. ACM Press. Sponsored by SIGACT: ACM Special Interest Group on Algorithms and Computation Theory.