Diagnosing Infeasibilities in Network Flow Problems

by

Charu C. Aggarwal
MIT Operations Research Center

Jianxiu Hao
GTE Laboratories

James B. Orlin
MIT Sloan School

Sloan School WP # 3696
First Draft: June 13, 1994

Diagnosing Infeasibilities in Network Flow Problems

Charu C. Aggarwal
Operations Research Center
Massachusetts Institute of Technology
MA 02139

Jianxiu Hao,
GTE Laboratories,
Waltham,
MA 02154

James B. Orlin,
Sloan School of Management,
Massachusetts Institute of Technology,
MA 02139

June 13, 1994

Abstract

We consider the problem of finding a feasible flow in which each node i has a supply b(i), and each arc has a lower bound of 0 on flow and an upper bound uij. It is well known that this feasibility problem can be transformed into a maximum flow problem. It is also well known that there is no feasible flow if and only if there is a subset S of nodes such that the sum of the supplies of the nodes of S exceeds the capacity of the arcs emanating from S. Such a set S is called a "witness to the infeasibility" of the network flow problem. In the case that there are many different witnesses for an infeasible problem, a small cardinality witness may be preferable in practice because it is generally easier for the user to assimilate, and may provide more guidance to the user on how to identify the cause of the infeasibility. Here we show that the problem of finding a minimum cardinality witness is NP-hard. We also consider minimal witnesses, that is, a witness S such that no proper subset of S is also a witness. The primary contribution of this paper is an algorithm for determining minimal witnesses whose running time is comparable to solving a single maximum flow problem.

Keywords: Network Flows, Infeasible, maximum flow problem

1 Introduction

An important area in the analysis of linear programming problems is that of diagnosing infeasibilities. In particular, given a set of linear inequalities, we would like to know whether this system is feasible or not. Further, once it is known that the system is infeasible, we would like to isolate the cause of the infeasibility even further: that is, we would like to find a small subsystem of equations and inequalities, which forms an infeasible subsystem. In general, it is NP-hard to find a minimum infeasible subsystem of equations for a linear programming problem; however it is possible to find an irreducible subsystem of equations which forms an infeasible subsystem in itself.[9], [3], [13]. (Irreducibility of a subsystem means that no subset of it is infeasible.)

In this paper we shall consider this problem for the network flow problem, which is a particular case of the linear programming problem. Diagnosing infeasibilities for network flow problems was first studied by Greenberg.([9] and [10]). The advantage in considering this particular case is that we can make use of the nice structure of the network flow problem in order to come up with fast combinatorial methods for solving it. Here, we consider network flow problems defined on a network $G = (N, A)$ with n nodes in N and m arcs in A. The supply/demand of node i is $b(i)$, where a positive value indicates a supply, and a negative number indicates a demand. The capacity of each arc (i, j) is u_{ij}, while the lower bounds on the arc flows is zero.[1] A *witness of infeasibility* is a subset S of nodes, which is such that the net supply of the set S is larger than the sum of the capacities of the arcs directed from S to $N - S$.[2] It is intuitively quite obvious that a network having such a property would not be feasible because the supply on the set S of nodes has to "escape" from S through the arcs directed from S to to N-S. In fact, Gale[6] proved that there is a feasible solution to a network flow problem if and only if there is no witness. In other words, in case the network flow problem has no solution, then this is a simple proof of infeasibility. Moreover, it is very easy to find *some* witness by solving a related maximum flow problem. (For more details, see for example, Ahuja et al[1].) While all witness are equally valid in proving that a network flow problem is infeasible, not all of them provide an equal amount of guidance to the user of a modeling system. This point is well articulated by Greenberg[10] who writes "I say that a [witness] offers a good diagnosis [if] the information from the [witness] provides a useful starting point, where we need only a modest amount of additional analysis to form a complete diagnosis that correctly identifies the cause."

One of the features of a witness that is particularly relevant to its use in a diagnosis is the cardinality of a witness. In general, the smaller the witness, the more easily a user can analyze it. The larger the witness, the more difficult it is for a user to comprehend its cause. In this paper, we shall focus on the criterion provided by Greenberg, the size of the witness. We say that a witness S is *minimal* if there is no subset S' of S which is also a witness. We say that a witness S is *minimum*, if there is no witness whose cardinality is less than that of S. In the next section, we shall show that it is NP-hard to determine a minimum cardinality witness. More properly, we shall show that it is NP-complete to determine whether there is a witness with at most k nodes.

The paper is organized as follows. In section 3, we give a simple algorithm for finding a min-

[1] As we shall see later, this is actually without loss of generality, because a network flow problem with nonzero lower bounds on arc flows can be easily converted to one whose lower bounds are all zero.

[2] This is actually a combinatorial characterization of an infeasible subsystem of equations.

imal witness as a sequence of n maximum flow problems. In Section 5, we shall show how to speed up the algorithm for finding a minimal witness , so that the running time is roughly equal to that of a single maximum flow problem; in this case the running time reduces to $O(nm \cdot log(n^2/m))$, which also happens to be the time it takes to solve a maximum flow problem using the Goldberg-Tarjan preflow push algorithm. We also use concepts and ideas contained in Hao and Orlin's minimum cut algorithm [11] in order to reduce the running time by a factor of n. This similarity in the time bound is not incidental; our algorithm heavily relies on the concepts derived from the Goldberg-Tarjan preflow push algorithm. Hence we make a brief presentation of this algorithm in Section 4. The correctness and the proof of the time bound is proved in Section 6, while a brief Conclusion and Summary is presented in Section 7.

2 NP-hardness of the minimum witness problem

In this section, we shall prove the NP-hardness of the minimum witness problem. More precisely, we shall show that it is NP-complete to determine whether a witness with k nodes exists. In order to prove this, we shall use the standard notion of reduction, as introduced by Cook in [4]. Further the problem from which we shall make the reduction is a special case of the k-clique problem, which is also known to be NP-complete. (See, for example, Cook [4]) We shall first prove the NP-completeness of this version of the clique problem, and then use it to prove that the k-witness problem is NP-hard as well. The revised problem is as follows:

The Clique Problem on Graphs of fixed degree:

Input: An undirected graph $G = (N, A)$ such that each node of N is incident to exactly d undirected arcs, and an integer k.

Question: Is there a clique in G with k nodes?

Lemma 1 *The clique problem on graphs of fixed degree is NP-complete.*

Proof: We shall prove this by reducing from the k-clique problem without restriction on the degrees of nodes, which is already known to be NP-complete. Suppose that we have an arbitrary graph $G = (N, A)$ and we wish to find whether a clique of size k exists. We shall transform this to an instance of the restricted degree clique problem. Let $d(i)$ be the number of arcs incident to node i in G, and let $d^* = max\{d(i) : i \in N\}$. We shall create a graph G' which is such that each node in it has degree $d^* + 2$, and any clique with $k \geq 3$ nodes is contained in G' if and only if it is contained in G.
Let $K = n(d^* + 2) - 2m$. Let G' be obtained from G by first appending a subset N' of K nodes forming a bipartite graph, in which each node has degree $d^* + 1$. (Observe that $nd^* \geq 2m$, and so $K \geq 2n$, and hence it is easy to ensure that each node in N' has degree $d^* + 1$.) Then, for each node i in G, create $d^* - d(i) + 2$ arcs from i to different nodes of N'. The number of arcs created in this way is K, since the sum of the node degrees is $2m$. The K additional arcs should be created in such a way that each node in N' has one additional incident arc. Subsequently, each node in the resulting graph G' has degree $d^* + 2$. Further since the subgraph formed on G' is bipartite, and

each arc from N to N' is directed to a distinct node in N', it is not difficult to see that a clique of size $k \geq 3$ exists in G if and only if it exists in G'. □

We shall now proceed to use this lemma in order to prove the NP-completeness of the k-witness problem. The k-witness problem may be formally stated as follows:

The k-witness problem:
Input: An undirected graph $G = (N, A)$, an integer k, a vector $b(.)$ of supplies, and a vector $u(.)$ of arc capacities.

Question: Is there a subset S of k nodes of N such that $b(S) > u(S, N - S)$ where $b(S)$ is the total supply value of the node set S, and $u(S, N - S)$ is the capacity of the cutset $(S, N - S)$?

Theorem 1 *The k-witness problem on undirected graphs is NP-complete.*

Proof: This problem is evidently in the class NP, because given a set of nodes we can easily check in polynomial time whether or not they form a witness set. Let d denote the fixed degree of a graph $G = (N, A)$ and suppose that we want to find out whether a clique of size k. We shall transform this to a k'-witness problem on an undirected network $G' = (N', A')$ with $k' = k + 1$. Let us now define the parameters of this problem instance more precisely:

1. Let $N' = N \cup \{s, t\}$.
2. Let $A' = A \cup \{(s, i) : i \in N\}$. The node t will have no incident arcs; that is, it is an isolated node.
3. The capacity of each arc in A is one unit.
4. The capacity of each arc (s, i) is d units.
5. $b(s) = -b(t) = nd - k(k - 1) + 1$; $b(i) = 0 \ \forall i \in N - \{s, t\}$.

The above network flow problem is obviously an infeasible one, because the sink node t is isolated and $b(t) < 0$. Note also that any witness of G' must contain the source node s, since it is the only node with a positive supply value. We now claim that G' has a witness of cardinality $k' = k + 1$ if and only if G has a clique of size k.
Suppose that G has a clique of size k, and let S be the set of nodes in the clique. Let $S' = S \cup \{s\}$. Then, $b(S') = nd - k(k - 1) + 1$. The number of arcs in the clique is $k(k - 1)/2$, and thus the number of arcs incident to a node in S as well as to a node in $N - S$ is $kd - k(k - 1)$. (Each node of S is incident to d arcs, and each arc of the clique is incident to two nodes in S.) Each of these arcs has a capacity of 1. The number of arcs incident to s and a node in $N - S$ is $n - k$, and the capacity of each of these arcs is d. It follows that the capacity of the cutset $(S', N' - S')$ is $kd - k(k - 1) + (n - k)d = nd - k(k - 1)$, and thus S' is a witness of cardinality k'.

Conversely, suppose that G' has a witness S' of cardinality $k' = k + 1$. As already stated above, $s \in S'$ and $t \notin S'$. Let $S = S' - \{s\}$, and let K be the number of arcs with both end points

in S. Then the capacity of the cutset is $nd - 2K$. Because S' is a witness this capacity is strictly less than $nd - k(k-1) + 1$; i.e. $nd - 2K < nd - k(k-1) + 1$. From this relationship, we can deduce that $K > k(k-1)/2 - (1/2)$. Thus, $K \geq k(k-1)/2$. But, the number of nodes with both end points in Scan be *at most* $k(k-1)/2$. Hence the number of arcs with both end points in S is exactly equal to $k(k-1)/2$, and hence S must be a clique in G. □

3 Overview of the Minimal Witness Algorithm

In the previous section we showed that it is NP-hard to find a witness of minimum cardinality. However, it is possible to find a compromise solution; that is, a *minimal* witness. Before we begin a detailed discussion of the algorithm, it is necessary to develop some additional concepts and notations.

One term that we need to define for this paper is a generalization of the concept of $s - t$ cut. Suppose that S and T are mutually disjoint sets of nodes. The minimum $S - T$ cut problem is the following:

$$\text{minimize } \{u(S^*, N - S^*) : S \subseteq S^*, T \subseteq N - S^*\}$$

In other words, we want a minimum cut subject to the additional restriction that the source side of the cut contains S, and the sink side of the cut contains T. Thus, we can view an $S - T$ cut in the network G as an $s - t$ cut in the network G', where G' is obtained from G by contracting all nodes in S to a single node s, and all nodes in T to a single node t.

Our algorithm draws concepts from the paper on the unrestricted minimum cut problem by Hao and Orlin[11]. We shall solve the minimal witness problem after imposing the following additional restriction:

Assumption 1 *The network $G = (N, A)$, in which the minimal witness has to be found, has exactly one supply node s and one demand node t. Any other node i has $b(i) = 0$.*

We shall now proceed to show that the above assumption is actually without loss of generality because it is always possible to transform an arbitrary network flow problem into one, which has exactly one supply node and one demand node.

Lemma 2 *A minimal witness problem in a network $G = (N, A)$ can be solved by solving a minimal witness problem in a corresponding network G' that contains exactly one source node and exactly one supply node.*

Proof: Let $G = (N, A)$ be the network in which the minimal witness problem is to be solved. Construct the network $G' = (N \cup \{s, t\}, A \cup A_0)$ where $A_0 = \{(s, i) : b(i) > 0\} \cup \{(i, t) : b(i) < 0\}$. Further the supply value of the source node s and the demand value of the sink node t is identically equal to $\sum_{i \in N, b(i) > 0} b(i)$ while all other nodes have zero supply/demand values. For each node i with $b(i) > 0$, the capacity of the arc (s, i) in G' is $b(i)$. Similarily, for each arc (i, t) on G', the capacity is equal to $-b(i)$. Then, it is easy to show that S is a witness set in G if and only if $S \cup \{s\}$ is a witness set in G' because the difference between the supply value of the corresponding set of

Algorithm *Find-Minimal-Witness(s,t)*;
begin
$S := \{s\}$; $T := \{t\}$; $TestNode := \phi$;
while $(S \cup T) \neq N$ **do**
begin
determine a minimum $S - (T \cup \{TestNode\})$ cut $[R, \overline{R}]$;
if R is a witness
then
add $TestNode$ as well as all sink side nodes to T;
else
add $TestNode$ to S;
Select a new node in $N - S - T$ to be the new $TestNode$;
end;
end;

Figure 1: A skeletal structure of the minimal witness algorithm

nodes and the total capacities of the arcs emanating from it is the same in both the cases. □

Our algorithm basically proceeds as follows: It maintains two disjoint node sets S and T, where S is a set of nodes which will be a subset of the final minimal witness S^* , while T is a set of nodes which will always be a subset of the complement of the witness set S^*. Initially $S = \{s\}$ and $T = \{t\}$. Assuming that the problem is indeed infeasible, the algorithm finds a minimum $S-T$ cut, $[R_0, \overline{R_0}]$, (hence R_0 is a witness set) selects an arbitrary node $i \in R_0$ and determines whether some subset of $R_0 - \{i\}$ is also a witness, by finding a minimum $S - [\overline{R_0} \cup \{i\}]$ cut and checking whether the node set thus obtained is a witness or not. If it is indeed a witness, it means that a witness which a subset of $R_0 - \{i\}$ exists, and hence we can add it to T; otherwise, we add this node to the set S. In the former case, we add all sink side nodes to T as well. We continue this procedure until $S \cup T = N$. Thus, a total of $O(n)$ iterations are required. since the size of either S or T increases in each iteration. Further, since each iteration requires the application of one maximum flow algorithm, the time complexity of this method is equal to that of the running time of O(n) maximum flow problems. However, as we shall see later, these $O(n)$ maximum flow problems are closely related to one another, and , as a result we can reduce the running time of these problems to that of a single application of the Goldberg-Tarjan preflow push algorithm. Thus, the minimal witness problem, when presented in its simplest form is as illustrated in Figure 1. We shall now prove that, at termination of the algorithm, the set S is a minimal witness set.

Theorem 2 *The algorithm Find-Minimal-Witness determines a minimal witness set.*

Proof: This property may easily be proved by observing that the algorithm maintains the invariant on the sets S and T, that $N-T$ contains some witness, and S is a subset of each witness contained in $N - T$. Hence, at termination when $S \cup T = N$, no proper subset of S can be a witness.
In order to prove that the algorithm indeed maintains the invariant stated by us above, we shall

note that we add TestNode to S if and only if $N - T - \{TestNode\}$ contains no witness. On the other hand, if a TestNode (and possibly other sink side nodes) are added to T when a witness $(R, N - R)$ is found, then subsequently T becomes $N - R$ and $N - T = R$ is a witness. Moreover, any witness which is a subset of R must also contain S, since this invariant was true at the previous iteration. □

The minimal witness algorithm solves the minimal witness problem as a sequence of $O(n)$ closely related minimum cut problems, and we exploit the closeness in order to reduce the total running time. that we can reduce the time complexity to that of a single maximum flow problem, if we are careful about the order in which we select our TestNodes. Since our method draws concepts from the Goldberg-Tarjan preflow push algorithm[8], we shall review this algorithm briefly in the next section. In subsequent sections, we shall show how to modify it so as to solve the minimal witness problem.

4 The Preflow Push Algorithm

In this section, we shall briefly describe the preflow push algorithm for finding the minimum $S - T$ cut, where S and T are mutually disjoint subsets of nodes. The algorithm is a modification of Goldberg and Tarjan's [8] preflow push algorithm for the maximum flow problem, and is also somewhat similar to the version of the preflow push algorithm described by Hao and Orlin in [11].We shall assume that the reader is already familiar with the Goldberg-Tarjan algorithm, and we shall focus on modifications of this approach.
The algorithm proceeds by maintaining an optimal *preflow* during each feasible step. A preflow is a flow in which the mass balance constraints are not necessarily satisfied, but the nodes may have nonnegative *excesses* because of the imbalances in the incoming and outgoing flow values. For each node $i \in N - S - T$, the excess is denoted by $e(i)$ and is defined as follows:

$$e(i) = \sum_{\{j:(j,i)\in A\}} x_{ji} - \sum_{\{j:(i,j)\in A\}} x_{ij} \quad \forall i \in N - S - T$$

The excesses are not defined for any of the nodes in the source set S or the sink set T. We say that x is a preflow if $0 \leq x_{ij} \leq u_{ij}$ for each arc (i, j) and $e(i) \geq 0$, for each node $i \in N - S - T$. where u_{ij} is the capacity of the arc (i, j). Given a flow/preflow x, we define the *residual capacity* of the node pair $[i, j]$, as $r_{ij} = (u_{ij} - x_{ij}) + x_{ji}$. The flow $u_{ij} - x_{ij}$ is the maximum additional increase on the arc (i, j), while the flow x_{ji} is the maximum amount by which the flow on the arc (j, i) can be decreased.[3] The *residual network* $G(x)$ with respect to the flow x, is defined to be the network $(N, A(x))$, where $A(x)$ is the set of all arcs with positive residual capacities.
The preflow push algorithm proceeds by partitioning the nodes into two parts, W and D , denoting the set of nodes which are *awake* and the set of nodes which are *dormant* respectively. At every stage of the algorithm, the property that $S \subseteq D$ and $T \subseteq W$ is maintained. A node i is said to be *active* if $i \in W - T$ and $e(i) > 0$. An arc (i, j) in $G(x)$ is said to be *admissible* if $i \in W$, $j \in W$, and $d(i) = d(j) + 1$. In general, the algorithm will send flow from active nodes and push it along admissible arcs.

[3] We are assuming here, for the purpose of abstraction, that whenever the arc (i, j) exists, so does the arc (j, i). This assumption is without loss of generality, because we can always add the arc (j, i) with zero flow and zero capacity.

The idea of identifying dormant and awake nodes was first presented as a heuristic for speeding up the Goldberg-Tarjan preflow push algorithm by Derigs and Meier [5]. At every stage, the algorithm maintains the property that there is no arc in the residual network from any node in D to any node in W. This property which the algorithm maintains throughout its execution is defined to be the *dormancy property.* In the initialization step, the algorithm sets $D = S$ and $W = N - S$, and saturates all the arcs emerging from the nodes in S, so that no arc exists in the residual network from any node in D to any node in W.

The algorithm performs pushes in a similar manner to the Goldberg-Tarjan preflow push algorithm, except that we never perform pushes either from or towards a dormant node. Similarily, the relabels performed are also similar to those of the Goldberg-Tarjan preflow push algorithm, except in two cases. In the first case a *gap cut* is found, that is, if a node i is to be relabeled, and there is no other node in W with distance label $d(i)$. In this case, the algorithm lets $R = \{j \in W : d(j) \geq d(i)\}$ and transfers the nodes of R from D to W. As we shall subsequently prove, the modified set D will continue to satisfy the property that no arc is directed from W to D. In the second case, when i is to be relabeled and no arc in $G(x)$ is directed from i to some node in W, we can add i to D and delete it from W. It is easy to see that no arc in the residual network will be directed from a node in $D \cup \{i\}$ to a node in $W - \{i\}$. In addition, the algorithm maintains the *W-validity property* as well, according to which, for each pair of nodes $i, j \in W$ if $r_{ij} > 0$ then $d(i) \leq d(j) + 1$. Hence, there can be no path in the residual network from a dormant node i to any node in W, and hence also to any node in $T \subseteq W$. In transferring nodes from W to D we do not modify them with distance labels. This subtle distinction between our algorithm and that of Goldberg and Tarjan is needed to improve the running time. The basic steps of the preflow push algorithm are illustrated in Figures 2, 3 and 4.
We shall now prove a result which is a variation of a lemma given in Goldberg and Tarjan [8], and is easily proved via induction.

Lemma 3 *Suppose that the set $d(.)$ of distance labels is W-valid, and the set $D = N - W$ satisfies the dormancy property. Then, for each node i, d(i) is a lower bound on the number of arcs in any path in $G(x)$ from node i to the node t.*

Proof: Omitted. See Goldberg and Tarjan. [8]. □

5 The Minimal Witness Algorithm

The minimal witness algorithm presented in this section uses the preflow push algorithm of the previous section (with some minor differences) as a subroutine. The primary difference lies in the relabel procedure which partitions the dormant set into subsets called $DormantSet(0)$, $DormantSet(1)$, ..., $DormantSet(D_{max})$ which satisfy the following *extended dormancy property:*

No arc exists from a Dormant set to the awake set W. Further, no arc exists from a lower indexed subset of D to a higher indexed subset; i.e. if $i < j$ then an arc (u, v) cannot exist, such that $u \in DormantSet(i)$ and $v \in DormantSet(j)$.

The dormant subsets are usually constructed by creating a new dormant subset $DormantSet(D_{max}+$

```
Algorithm PreflowPush(S,T);
begin
  Initialize;
  while the network contains an active node do
  begin
    Select an active node i;
    if the network contains an admissible arc (i,j) then
      push δ := min{e(i), r_ij} units of flow from node i to node j;
    else
      relabel(i);
  end;
end;
```

Figure 2: The Preflow Push Algorithm

```
Procedure Initialize;
begin
  For each arc (i,j) with i ∈ S and j ∈ N − S, send r_ij units of
        flow on the arc (i,j);
  D := S;
  W := N − S;
  Let d(t) := 0 for each node t ∈ T;
  for each node j ∈ N − T do d(j) := 1;
end;
```

Figure 3: The procedure *Initialize*

Procedure *relabel(i)*;
begin
if i is the only node in W with distance label d(i) **then**
begin
$R := \{j \in W : d(j) \geq d(i)\}$;
$D := D \cup R$;
$W := W - R$;
end
else if there is no arc (i, j) in G(x) with $j \in W$,
then
$D := D \cup \{j\}$ and $W := W - \{j\}$;
else
$d(i) := min\{d(j) + 1 : (i, j) \in A(i), j \in W \text{ and } r_{ij} > 0\}$;
end;

Figure 4: The Relabel Procedure

1) whenever nodes are transferred from W to D. At the same time, we also increment D_{max} by one.
As described in a previous section, the algorithm finds an $S - T$ cut in each iteration. In the subsequent iteration, it finds an $S - (T \cup TestNode)$ cut. Then it transfers $TestNode$ to either set S or set T, depending on certain conditions which we shall describe later. In the case that TestNode is transferred to T, all other awake nodes are transferred to T as well. It may be noted that since all the awake nodes are also transferred to T in one case, the algorithm will empirically turn out to be very efficient. Unlike the minimum cut algorithm of Hao and Orlin, the inner loop will (in practice) be repeated fewer than n times. The algorithm selects TestNode by choosing the node in W-T with the minimum distance label. (In the case that $W - T$ is empty, the algorithm transfers $DormantSet(D_{max})$ to W and decrements D_{max} by one.) A detailed description of the algorithm is given in Figures 5, 6, 7 and 8.

As we have already claimed before, the transfer of a set of nodes R from W to T during the execution of the procedure relabel does not affect the dormancy property. We formally state the result as follows:

Lemma 4 *Suppose that the set d(.) of distance labels is W-valid, the set D=N-W satisfies the dormancy property, $i \in W - T$ has no admissible arcs emanating from it, and there is no other node in W-T with distance label d(i). Let R:=$\{j \in W - T : d(j) \geq d(i)\}$. Then there is no arc in the residual network which is directed from a node in R to a node in W-R. In addition, D' satisfies the extended dormancy property.*

Proof: By assumption and by the definition of the set R, there is no arc directed from node i to a node in $W - R$. Let j be any node in R and k be any node in $W - R$. By definition of the set R, $d(j) > d(i) > d(k)$; that is, $d(j) \geq d(i) + 1 \geq d(k) + 2$. Therefore, $d(j) \geq d(k) + 2$. It follows from

```
Procedure Minimal Witness;
begin
  Initialize
  while S ∪ T ≠ N do
  begin
    while W − T − {TestNode} contains an active node do
    begin
      Select an active node i;
      if the network contains an admissible arc (i, j)
        then
        push δ = min{e(i), r_ij} units of flow from node i to j.
        else
        Relabel(i));
    end;
    Test-Another-Node;
  end;
  Minimal − Witness − Set := S;
end;
```

Figure 5: The Minimal Witness Algorithm

```
Procedure Initialize;
begin
  for each arc (s, j) send r_sj units of flow in (s, j);
  S := {s};T := {t}; TestNode := φ;
  DormantSet0 := {s};
  D_max := 0;
  W := N − {s};
  d(t) := 0;
  for each node j ∈ N − {t} do d(j) := 1;
end;
```

Figure 6: The procedure *Initialize*

Procedure *relabel(i)*;
begin
if i is the only node in W with distance label d(i) **then**
begin
$R := \{j \in W : d(j) \geq d(i)\}$;
$D_{max} := D_{max} + 1$;
$DormantSet(D_{max}) := R$;
$W := W - R$;
end;
else if there is no arc (i, j) in $G(x)$ with $j \in W$,
then
$D := D \cup \{j\}$ and $W := W - \{j\}$;
else
if there is no arc (i, j) in $G(x)$ with $j \in W$
then begin
$D - max := D_{max} + 1$;
$DormantSet(D_{max}) := \{i\}$;
$W := W - \{i\}$;
end
else
$d(i) := min\{d(j) + 1 : (i, j) \in A(i), j \in W \text{ and } r_{ij} > 0\}$;
end;

Figure 7: The Relabel Procedure

```
Procedure Test-Another-Node;
begin
  /* TestNode is the node which is currently being tested */
  if u[D, N − D] ≥ b(s)
    then
    add TestNode to S and DormantSet(0)
    /*Note that S is always maintained to be the same as DormantSet(0)*/
    else
    T := T ∪ W;
  if S ∪ T = N then quit else continue;
  if TestNode has been added to S then
     saturate each arc from TestNode to a node in N − S;
  if W − T = φ then
  begin
    W := DormantSet(D_max) ∪ T;
    D_max := D_max − 1;
  end;
  select j ∈ W − T such that d(j) is minimum;
  Let j be the new TestNode;
  Set distance label of every node in T equal to that of TestNode;
end;
```

Figure 8: Procedure *Test-Another-Node*

W-validity conditions that the arc (j, k) cannot exist. Hence there is no arc directed from a node in R to a node in $W - R$, and consequently no arc is directed from a node in $D \cup R$ to a node in $W - R$. □

6 Correctness of Algorithm, TimeBounds and Implementation Issues

The proof of correctness of the algorithm and analysis of time bounds is slightly long and considerably involved. It requires us a prove quite a few intermediate lemmas before we can prove the entire result. Hence, we shall first state all the intermediate lemmas in one place, so that the reader may get an overview of the flow of logic before getting to a final conclusion. The various intermediate results that we shall prove are the following:

(1) Suppose that S and T denote the source and the sink nodes respectively. If there are no active nodes in the current preflow x^0, then $[D, N - D]$ is a minimum capacity $S - T$ cut. (This result, in conjunction with Theorem 2 guarantees correctness.)

(2) At each stage of the algorithm, no arcs exist from $W - T$ to T.

(3) W-validity is maintained throughout the execution of the algorithm.

(4) The distance label of each node i is nondecreasing throughout the execution of the algorithm, as long as it does not enter T.

(5) There is no arc of the residual network directed from a node in D to a node in W. (Dormancy Property). Further, for any pair of indices i, j such that $0 \leq i < j \leq D_{max}$ there is no arc of the residual network directed from $DormantSet(i)$ to $DormantSet(j)$.

(6) d(W-T) is a set of consecutive integers. Suppose that $R = DormantSet(j)$ for some j. Then $d(R)$ is a set of consecutive integers.

(7) Each node label is increased at most $(n - 1)$ times before it enters T. In fact, $d(j) \leq (n - 1)$ for all the nodes at all iterations.

(8) Each arc is saturated at most $(n - 1)$ times. Hence there are a total of at most $O(nm)$ saturating pushes.

(9) The number of times that some set of nodes is transferred from D to W is at most n. The number of times that a set can be transferred from W to D is at most $2n$.

(10) The algorithm *Minimal-Witness* requires a time of O(nm + number of nonsaturating pushes).

(11) The algorithm *Minimal-Witness* with highest level pushing runs in time $O(n^2 \cdot m^{0.5})$.

(12) The algorithm *Minimal-Witness* with FIFO pushing runs in time $O(n^3)$.

(13) The dynamic trees implementation as described in Goldberg and Tarjan [8] requires $O(nm \cdot log(n^2/m))$ steps.

We shall now get down to proving each of these individual items sequentially.

Lemma 5 *Suppose that S and T denote the source and the sink nodes respectively. If there are no active nodes in the current preflow x^0, then $[D, N-D]$ is a minimum capacity $S-T$ cut.*

Proof: Any preflow x^0 can be converted into a flow x without affecting the total flow into the sink node T. (See, for example, Goldberg and Tarjan [8].) Further, since no node in $N-D-T$ has any excess, the total flow due to x^0 across the cut $(D, N-D)$ is equal to its capacity, and the amount of flow that x^0 brings into T is equal to the value of the flow x. Thus the flow x is maximum and the cut $(D, N-D)$ is minimum. This follows directly from the duality of maximum flow and minimum cut, since we have found a feasible flow x and a feasible cut $[D, N-D]$, such that their objective function values are equal. □

Lemma 6 *At each stage of the algorithm, no arcs exist from nodes in $W-T$ to nodes in T.*

Proof: Instead of proving the above result, we shall prove the much stronger property that at each stage of the algorithm, there are no arcs directed from nodes in $N-T$ to nodes in T. The property is trivially true at the beginning of the algorithm because the algorithm starts with $T = \phi$. Further, the only time when the set of arcs between T and $N-T$ could change is when the set T is changed. (The set T is changed during all those steps of the algorithm when we add the current $TestNode$ to T.) However, whenever we add $TestNode$ to T, we also add all the nodes which lie on the sink side of the minimum cut to T. By definition of the minimum cut, the residual network cannot contain an arc directed from the sink side of the cut to the source side. As a result, the above property is maintained. □

Lemma 7 *W-validity is maintained throughout execution of the algorithm.*

Proof: The initialization step starts off with W-valid distance labels. It is easy to prove further that distance labels remain W-valid following a push or a relabel. (See, for example, Goldberg and Tarjan [8]). It remains to prove that the resetting of distance labels of nodes in T in the last line

of the execution of the procedure *Test-Another-Node* does not affect W-validity. Since we already know from the previous lemma that no arcs exist from nodes in $W - T$ to nodes T, such a resetting could only affect the W-validity of an arc from T to $W - T$, if at all. However, the distance label of each node in T is set to a value which is no larger than the distance label of any node in $W - T$. (Actually, the distance label of each node in T is set to be equal to the minimum distance label in $W - T$, since *TestNode* was selected to be the node with minimum distance label in $W - T$.) Hence, any arc which is directed from $W - T$ to T will remain W-valid even after the resetting of distance labels in T. □

Lemma 8 *The distance label of each node i is nondecreasing throughout the execution of the algorithm as long as it does not enter T.*

Proof: It follows from W-validity conditions that there is never any arc with $d(i) > d(j)+1$. Thus, there is never any need to reduce the distance label of a node. (The only step at which the distance label of a node *might* be decreased is in the execution of the procedure *Test-Another-Node* when the distance labels of all the nodes in T are reset.) □

Lemma 9 *There is no arc of the residual network directed from a node in D to a node in W. (Dormancy Property). Further, for any pair of indices i,j such that* $0 \leq i < j \leq D_{max}$ *there is no arc of the residual network directed from DormantSet(i) to DormantSet(j). (Extended Dormancy Property).*

Proof: At the end of the initialization step, the dormancy/extended dormancy conditions hold. It remains to prove that these conditions are maintained during the execution of various steps of the algorithm. Transfer of nodes from D to W does not cause dormancy/extended dormancy conditions to be violated, if these conditions were satisfied before the transfer. Pushes do not affect the dormancy property because pushes take place only on arcs whose both head as well as tail nodes lie in W. In the case when the relabel procedure is called and no nodes are transferred to D, the conditions will not be violated. In the event that some nodes are transferred to D, it follows directly from the discussion in Lemma 4, that dormancy conditions will continue to be satisfied. Extended dormancy conditions will also be satisfied because dormancy as well as extended dormancy conditions were satisfied just before the transfer. □

Lemma 10 *d(W-T) is a set of consecutive integers. Suppose that R=DormantSet(j) for some j. Then d(R) is a set of consecutive integers.*

Proof: It is easy to see that d-consecutiveness properties hold at the end of each initialization step. It remains to prove that this property will not be affected by the execution of the various steps of the algorithm. A push certainly does not affect d-consecutiveness, because pushes do not change distance labels. The relabel operation does not affect d-consecutiveness, because we do not change the distance label of a node if it is the only node having that label. In the case that we require to transfer nodes during an execution of the *Relabel* procedure, consecutiveness of the labels is maintained, because a new dormant set is created out of some of the nodes in W in such a way that the d-consecutiveness of both the new dormant set and the remaining nodes in W is maintained. It remains to prove that the execution of the procedure *Test-Another-Node* does not

affect d-consecutiveness. This is not difficult to see because the only case in which the execution of the procedure could be suspected to violate d-consecutiveness is when nodes are transferred from D to W. However, in this case when $DormantSet(D_{max})$ is transferred to W, d-consecutiveness of the set $W - T$ is maintained because $W - T = \phi$ before the transfer and R is also d-consecutive. □

Lemma 11 *Each node label is increased at most* $(n-1)$ *times before it enters* T. *In fact,* $d(j) \leq (n-1)$ *for all the nodes at all iterations.*

Proof: Define $d_{min}(R) = min\{d(i) : i \in R\}$ and $d_{max}(R) = max\{d(i) : i \in R\}$. We claim for each dormant set R that $d_{min}(R) \leq n - |R|$ and $d_{min}(W - T) \leq n - |W - T|$. Since $d(R)$ and $d(W - T)$ satisfy the d-consecutiveness property, it implies that $d_{max}(R) \leq d_{min}(R) + |R| - 1$. Similarily, $d_{max}(W - T) \leq d_{min}(W - T) + |W - T| + 1 \leq n - 1$. Thus the distance label of each node is no larger than $(n-1)$ units. Since the distance labels are nondecreasing as long as they do not enter T, it follows that each node label is increased at most $(n-1)$ times before entering T.
These claims are easily seen to be valid after the initialization step, when $|W - T| < n$ and $d(t) = 0$ for the sink node t. It now remains to prove that the upper bound on the distance labels is not violated because of the various operations of the algorithm. A push or a modification of a distance label in $W - T$ does not change $d_{min}(W - T)$. Further, transferring the nodes from W to D in a relabel operation decreases $|W|$ but does not affect $d_{min}(W)$, and so, the property $d_{min}(W) \leq n - |W|$ remains true subsequent to the operation. In the case when the only one node i is transferred from W to D in a relabel operation, $DormantSet(D_{max})$ will contain a single node whose distance label is at most (n-1). On the other hand, in the case when $R = \{j \in W : d(j) \geq d(i)\}$, D_{max} is incremented and $DormantSet(D_{max})$ is set equal to R. In this case, $d_{min}(R) = d(i)$. But since $d(W - R)$ is consecutive, it follows that $d(i) \leq d_{min}(W) + |W - R| \leq n + |W - R| - |W| = n - |R|$. Hence the property holds for the new dormant set $DormantSet(d_{max})$ after the transfer. □

Lemma 12 *Each arc is saturated at most* $(n-1)$ *times. Hence there is a total of at most O(nm) saturating pushes.*

Proof: For each arc (i, j) there is at most one saturating push between consecutive relabels of a node i. Since a push can take place only from a node in $W - T$, and the number of distance label updates (as long as it does not enter T) is less than n, it follows that the number of times any arc (i, j) can be saturated is less than n. (This is because each pair of successive saturations in opposite directions on an arc increase the distance labels by two units for each of the endpoints.) Hence the number of arc saturations in total can be at most $O(nm)$. □

Lemma 13 *The number of times that some set of nodes can be transferred from D to W is at most n. The number of times that some set can be transferred from W to D is at most 2n.*

Proof: Nodes are transferred from D to W only when $W - T = \phi$ and we are executing the procedure *Test-Another-Node*. Since at most $(n-1)$ nodes are tested, the total number of transfers is also bounded above by $(n-1)$. As far as transfers of nodes from W to D are concerned, at most $(n-1)$ transfers take place due to additions of old TestNodes to S.Further, the number of transfers

due to the creation of new dormant sets is at most $(n-1)$ as well because each of these dormant sets will ultimately be transferred back to W in an execution of the *Test-Another-Node* procedure. □

So far, we have proved the various intermediate results necessary in order to obtain the time complexity of the algorithm. It now remains to prove to put these results together, so that we can find out the exact time complexity of the algorithm, which happens to be implementation dependant. We will note that so far, we have not commented on the time complexity of the number of nonsaturating pushes required, which is usually the bottleneck operation. The analysis required for the remaining section is so similar to that presented elsewhere, that we shall review them briefly here and give the references where a more detailed analysis may be found. In selecting admissible arcs, over a which a push may take place, we use the "current arc data structure" as described in Goldberg and Tarjan.[8] In this data structure, we shall use a pointer call $CurrentArc(i)$ which points to one of the arcs in $A(i)$. Whenever we search for an admissible arc, $CurrentArc(i)$ is incremented, until it eventually points to an admissible arc, or until it discovers that there is no admissible arc emanating from that node. In the latter case, the procedure $relabel(i)$ is called. The amount of time spent in scanning the arcs emanating from the node i is equal to $A(i) \cdot$ (Number of relabels of node i) $\leq n \cdot |A(i)|$. Hence the total time spent in scanning all the nodes $= O(n \cdot \sum_{i \in N} A(i)) = O(nm)$.
As far as the data structures for the node selection operations are concerned, one of the alternatives is to maintain a set of (n-1) lists $Active(k)$ which represents the set of active nodes whose distance label is k. These sets can easily be maintained at an additional cost of $O(1)$ per push, relabel and node-transfer. (This data structure is especially useful when we perform node selection in such a way that we push from the node with the highest distance label.) Further, we maintain a list called $numb(k)$, which denotes the number of nodes whose distance label is k. This array is also maintained at an additional cost of $O(1)$ per operation. Thus the running time of the algorithm is due to the following steps:

(1) **The time for initialization:** This requires $O(m)$ time.

(2) **Time for node selection:** Each time we select an active node, we perform either a push or a relabel. Hence the time required is at most equal to the $O(n^2 +$ number of pushes$) =$ $O(nm +$ number of nonsaturating pushes$)$. This result assumes that each node selection can be performed in O(1) time which is the case when we use the data structures discussed above. (For more details, see Ahuja et. al.[1].

(3) **Time spent for selecting admissible arcs:** As we have already established, this requires $O(nm)$ time.

(4) **Time spent for saturating pushes:** As we have already established, this requires $O(nm)$ time.

(5) **Time spent for non-saturating pushes:** This is implementation dependant. In the case we use the **highest label** implementation, (i.e. select a node whose distance label is as high

as possible) the number of nonsaturating pushes is $O(n^2 \cdot m^{0.5})$. In the case of the FIFO implementation, we require a total of $O(n^3)$ nonsaturating pushes. We omit the details of these proofs as they can be found in [1] and [2].

(6) **Time spent for increasing the distance labels during the execution of *relabel(i)*:** Since each node is relabeled at most n times, this requires at most $O(n^2)$ time.

(7) **Time for creating new dormant sets or transferring back dormant sets to W:** Since at most $O(n)$ such transfers can take place and each transfer requires $O(n)$ time, the total time required is $O(n^2)$.

(8) **Time for selecting a new TestNode:** Since a TestNode is selected at most $O(n)$ times and each selection takes at most $O(n)$ time for any implementation, this operation can be accomplished in time $O(n^2)$ (and, in fact, we can do much better in most cases.)

(9) **Time spent for relabeling the nodes in T:** The nodes in T can be relabeled only when a new node is being tested. This is done at most $O(n)$ times. Further, since each such operation requires $O(n)$ time, the total time required is $O(n^2)$.

By summing up the times for the various steps above, we obtain the following results:

Theorem 3 *The procedure Minimal-Witness determines the minimal witness in time $O(nm +$ Number of nonsaturating pushes$)$.*

Theorem 4 *The algorithm Minimal-Witness with highest level pushing runs in time $O(n^2 \cdot m^{0.5})$.*

Theorem 5 *The algorithm Minimal-Witness with FIFO pushing runs in time $O(n^3)$.*

Theorem 6 *The dynamic trees implementation as described in Goldberg and Tarjan [8] requires $O(nm \cdot log(n^2/m))$ steps.*

Proof: Since the transfer of nodes between D and W is not the bottleneck operation (and this is the only difference from the analysis presented by Goldberg and Tarjan [8]), the algorithm *Minimal-Witness* runs in time $O(nm \cdot log(n^2/m))$. We refer the reader to refer to [8] for additional details. □

7 Conclusion

We have shown that the problem of determining a minimum cardinality witness is NP-hard. In addition, we have shown that the problem of determining a minimal witness can be solved as a sequence of n maximum flow problems, and that the total running time of these n problems is O(nm log n2/m), which is comparable to the time to solve a single maximum flow problem.

The primary contribution of this paper has been theoretical, that is, the improved worst case running time for finding a minimal witness. Nevertheless, we are optimistic that the algorithm would perform well in practice. Perhaps the algorithm of this paper could be a core component of a system that permitted users to explore different witnesses, with the user specifying certain subsets to be included in or excluded from the witness.

The issue of identifying minimal witnesses also raises questions concerning the interpretation of the witness in the context of the model and the correction of the infeasibilities. Here we only raise the issue since it is well beyond the scope of this paper. Greenberg [1993], who uses the word "isolation" rather than "witness," writes: "An isolation is a portion of the linear program obtained in some purposeful way to contain a probable cause. A diagnosis [of the infeasibility] additionally requires an explanation of an isolation, which can require complex reasoning." We refer the reader to Greenberg [1993] for further discussion on the use of witnesses in diagnoses.

Acknowledgements

This research was partially supported by ONR contract N00014-94-1-0099 and also a grant from the UPS foundation.

References

[1] Ahuja R. K. , Magnanti T. L. and Orlin J. B. 1993 Network Flows:Theory Algorithms and Applications, *Prentice Hall, Englewood Cliffs* .

[2] Cheriyan, J. and S. N. Maheshwari. 1987. *Analysis of Preflow Push Algorithms for Maximum Network Flow.* Technical report, Department of Computer Science and Engineering, Indian Institute of Technology, New Delhi, India.

[3] Chinneck, J. W. and Dravnieks, E. W. 1991. *Locating minimal infeasible constraint sets in linear programs.* ORSA Journal of Computing, Vol 3, No. 2, pp 157-168.

[4] Cook S. A. 1971 *The Complexity of Theorem Proving Procedures.* Proceedings of the 3rd Annual ACM Symposium on Theory of Computing.

[5] Derigs, U., and W. Meier. 1988. *Implementing Goldberg's MaxFlow Algorithm: A Computational Investigation.* Technical Report, University of Bayreuth, West Germany.

[6] Gale D. 1957. *A theorem on flows in networks.* Pacific Journal of Methematics 7, 1073-1082.

[7] Gallo, G., M. D. Grigoriadis, and R. E. Tarjan. 1989. *A Fast Parametric Flow Algorithm.* SIAM Journal of Computing, 18, 30-55.

[8] Goldberg, A.V., and R.E. Tarjan. 1988. *A New Approach to the Maximum Flow Problem.* Journal of the ACM 35, 921-940.

[9] Greenberg, H. J. 1988. *Diagnosing infeasibility for min-cost network flow models, part II: primal infeasibility,* IMA Journal of Mathematics in Business and Industry 4, 39-50.

[10] Greenberg, H. J. 1993. *How to Analyze the results of Linear Programs – Part 3: Infeasibility Diagnosis.* Interfaces 23, 120-139.

[11] Hao J. and Orlin J.B. 1992. *A Faster Algorithm For Finding The Minimum Cut in a Graph.* Proceedings of the 3rd Annual ACM-Siam Symposium on Discrete Algorithms. pp 165-174. (Expanded version accepted for publication by the *Journal of Algorithms*

[12] Padberg, M. and G. Rinaldi. 1990. *An Efficient Algorithm for the Minimum Capacity Cut Problem.* Math. Programming 47, 19-36.

[13] Van Loon, J. N. M. 1981. *Irreducibly Inconsistent Systems of Linear Inequalities.* European Journal of Operations Research. Vol. 8, pp. 283-288.