Preprocessing in Stochastic Programming:
The Case of Uncapacitated Networks

STEIN W. WALLACE  Haugesund Maritime College, Skåregaten 103, N-5500 Haugesund, Norway
INTERNET: sww%hmb@gran.ii.uib.no

ROGER J-B WETS  Department of Mathematics, University of California, Davis, Davis, CA 95616
ARPANET: jbroger@violet.berkeley.edu

(Received: November 1988, final revision received: May 1989, accepted: June 1989)

Preprocessing can speed up the solution procedures for two-stage stochastic programming. We consider the case when the second-stage problem is a pure, uncapacitated network. We describe a number of procedures to reduce the size of the recourse problem. We describe a procedure for generating efficiently the (induced) feasibility cuts, and show that further reductions are possible if more information about the node-types is taken into account. We also investigate network collapsing techniques that would simplify the work required to find both optimality cuts and feasibility cuts, if we had not yet reduced the problem to one with relatively complete recourse. Computational results confirm that substantial savings are possible.

In the modeling of problems whose motivation comes from economic planning or resource management one must usually deal with uncertainties of many kinds. Typical examples would be uncertainty about future prices, uncertainty about the demand for certain commodities, etc. An example from fisheries management can be found in Wallace\cite{24} where there is uncertainty about the recruitment of the fish stock. The problem is to develop a long-term plan for the location of fish meal plants in Norway, using the fact that fishing quotas vary substantially from one year to the next, often with as much as 50--100%. When determining the long-term optimal layout, it is important to take into account that the fishing vessels will adjust to the varying fishing quotas, and thereby affect the amount of resources available to the individual plants. Technically speaking, the above problem is a two-stage stochastic mixed integer programming problem where the second-stage problem (catching the fish at the appropriate fishing ground and taking it to the closest point with available processing capacity) is an uncapacitated network flow problem.

Due to limitations in the ability to solve stochastic problems, decision makers often resort to deterministic models even if in many cases the results may not only be very far from a "reasonably" optimal decision, but may in certain cases be seriously misleading. The purpose of this paper is mainly to tighten the gap between the ability to formulate and the ability to solve stochastic optimization problems. We note, however, that the results on facet generation are significant in their own right.

The theory developed for linear stochastic pro-

grams with recourse suggests a number of shortcuts that could be used in the design of solution procedures or in the search for approximate solutions. Much of this has to do with the way feasibility can be checked, but in certain cases, as we shall see, it is possible to go much beyond that. Taking full advantage of these techniques usually requires problem reformulation: adding induced constraints, deleting redundant constraints, calculating "extremals" for the support of the random elements, etc. Because most of this work needs to be, and can be, done before the start of the solution procedure per se, we refer to it as preprocessing.

This paper reports on how preprocessing could help, sometimes dramatically, in the solution of two-stage stochastic programs with recourse when the second-stage problem is a pure uncapacitated network flow problem. We shall be concerned with three different possibilities: problem reduction by removal of arcs and nodes (in the second-stage network), adding induced constraints to obtain relatively complete recourse, and simplification of the feasibility check if relative complete recourse turns out to be too difficult or too time consuming to achieve.

The stochastic programming problem under consideration can be formulated as follows.

$$\min_x qx + \mathcal{E}(x)$$

subject to $Ax = a$, $x \geq 0$

where

$$\mathcal{E}(x) = E[Q(x, b, d)] = \int Q(x, b, d) \, dP$$

Subject classifications: Programming; preprocessing in stochastic programming, Networks: description of feasibility using facets.
and
\[ Q(x, b, d) = \inf_y \{ dy \mid Ey = H_y b + H_x y, y \geq 0 \}. \]

Here \( q, A, a, E, H_1 \) and \( H_2 \) are given vectors and matrices, and \( b \) and \( d \) are (possibly) random with probability distribution \( P \). The function \( Q(x, b, d) \) is called the recourse function. It is convex in \( b \) and concave in \( d \). The matrix \( E \) is the node-arc incidence matrix for the network with random arc costs \( d \) and with random supplies and demands determined by \( b \).

We assume that the reader is familiar with the L-shaped method (and/or its regularized version proposed by Ruszczynski\cite{Rusz}) for solving stochastic programs with recourse. For a review of the issues that need to be addressed in the implementation of this method, one could consult the articles by Birge\cite{Birge} and Wets\cite{Wets}. A simple description is given in Appendix C. It is well known that as long as some weak (not restrictive in practice) summability conditions are satisfied by the random quantities \( (b, d) \), that \( x \) is feasible if and only if

\[ A \hat{x} = a, \quad \hat{x} \geq 0 \quad \text{and} \quad E^* H_2 \hat{x} \leq e^* \]

where \( E^* \) is the polar matrix of \( E \), and for all \( j \)

\[ e^*_j = \sup_{b \in \text{suppb}} \{ E^* H_1(j) b \}, \]

with \( \text{suppb} \) the support of \( b \) (see Wets\cite{Wets}) and \( H_1(j) \) the \( j \)th column of \( H_1 \). In general calculating \( E^* \)—whose rows essentially correspond to the facets of \( \text{pos} E = \{ t \mid Ey = 0 \} \)—is a common definition of the polar matrix—could be prohibitively time consuming. In Section 2 we derive a number of results that, in our setting, allow us to do this with relative ease, and this is confirmed by our numerical results detailed in Section 3. The next section is devoted to the check for second-stage feasibility if we were not able to generate \( E^* \). The two final sections are concerned with network collapsing techniques (which in the case of random costs yield bounds on the stochastic programming problem).

In order to make the exposition simpler, we shall assume that \( \text{suppb} \) is bounded; when we refer to the extremal elements in \( \text{suppb} \) we mean the extreme points of the convex hull of \( \text{suppb} \).

Before we proceed, we need to review some terminology and notation that is of common use in the network literature, see in particular Rockafellar\cite{Rock}. Consider a connected, uncapacitated network with node set \( \mathcal{N} = \{ 1, 2, \ldots, n \} \). Arcs will be referred to as pairs of nodes. If arc \( k \) goes from node \( i \) to node \( j \) we write \( k \sim (i, j) \), and \( i = O(k), j = T(k) \). There can be several arcs between nodes \( i \) and \( j \).

When a network has random supply or demand, a slack node is needed. The external flow at the slack node will be chosen such that the total external flow is zero. We assume that node \( n \) is the slack node. Throughout the paper we denote by \( \beta(i) \) the external flow at node \( i \). If \( \beta(i) > 0 \) then \( i \) is a supply node, and if \( \beta(i) < 0 \) then \( i \) is a demand node.

An arc \( k \) where the support of \( d(k) \), denoted \( \text{suppd}(k) \), is not a singleton set will be referred to as a random arc. 1. Facet Generation

The purpose of this section is to present an algorithm for finding all facets of \( \text{pos E} \), where \( E \) is the node-arc incidence matrix of the network with row \( n \) that corresponds to the slack node) deleted, and

\[ \text{pos } E = \{ t \mid t = Ey, y \geq 0 \}. \]

The results of this section are of a general nature. In our context, we would first perform some network reductions to be discussed in a later section. Let

\[ F(i) = \{ \text{arcs } k \mid O(k) = i \} \]
\[ B(i) = \{ \text{arcs } k \mid T(k) = i \}. \]

We shall often refer to a related pair of sets that designates all the arcs that "follow" or are "before" node \( i \) or a set of nodes \( Y \). Let

\[ F^*(Y) = \{ \text{nodes } j \mid \text{there is a directed path from some node } i \in Y \text{ to node } j \} \cup Y \]
\[ B^*(Y) = \{ \text{nodes } j \mid \text{there is a directed path from node } j \text{ to some node } i \in Y \} \cup Y. \]

If \( Y = \{ i \} \), we shall write \( F^*(i) \) and \( B^*(i) \) instead of the more cumbersome \( F^*(\{ i \}) \) and \( B^*(\{ i \}) \). The sets \( F^*(i) \) can be found using the following procedure. (A similar procedure exists for \( B^*(i) \).) The procedure solves, in an efficient way, what is normally referred to as the "transitive closure problem," see e.g. Horowitz and Sahni\cite{Horow}. The computational complexity is \( o(n^3) \).

```plaintext
procedure CreateFstar;
begin
for all \( i \in \mathcal{N} \) do
  \[ F^*(i) := \bigcup_{k \in F(i)} \{ T(k) \} \cup \{ i \}; \]
for \( k := 1 \) to \( n \) do
  for \( i := 1 \) to \( n \) do
    if \( k \in F^*(i) \) then \( F^*(i) := F^*(i) \cup F^*(k) \);
end;
```

It is important to be aware of the implications of describing the network using \( F^*(i) \) and \( B^*(i) \), rather than \( F(i) \) and \( B(i) \). In some sense, \( F^*(i) \) and \( B^*(i) \) allow us to view the network in terms of another network with a minimal number of arcs, but with the
same connections. This "minimal network" is very
important for results later in this section, in particular
for Propositions 13–18. An example in Appendix A
illustrates how transitive closure affects the problem at
hand.

Let \( Y \subseteq \mathcal{N} \), and define

\[
Q^* = [Y, \mathcal{N} \setminus Y]^* = \{k \sim (i, j) | i \in Y, j \in \mathcal{N} \setminus Y\}
\]

\[
Q^- = [Y, \mathcal{N} \setminus Y]^* = \{k \sim (i, j) | j \in Y, i \in \mathcal{N} \setminus Y\}.
\]

We call \( Q = Q^+ \cup Q^- \) a cut.

From results in Gale\(^{10}\) and Hoffman\(^{11}\) we can
easily deduce the following result that can be used to
identify facets of pos \( E \).

**Proposition 1.** An uncapacitated network flow problem
is feasible if and only if for all node-sets \( Y \) where the cut
\( Q = [Y, \mathcal{N} \setminus Y] \) is such that \( Q^* = \emptyset \) there is no flow
from \( Y \) to \( \mathcal{N} \setminus Y \).

A straightforward (but computationally cumbersome)
way of finding all the facets of pos \( E \) would therefore be to use the following recursive procedure.
Initially it should be called with \( Y = \mathcal{N} \) and \( W = \emptyset \).

```
procedure BasicFacets(Y, W; set of nodes);
begin
  if \( Y \neq \emptyset \) then begin
    PickNode(i, Y);
    Y := Y \{i\};
    BasicFacets(Y, W);
    W := W \{i\};
    if \([W, \mathcal{N} \setminus W]^* = \emptyset\) then CreateIneq(W);
    BasicFacets(Y, W);
  end;
end;
```

This corresponds to full enumeration of all sets \( W \subseteq \mathcal{N} \). The procedure PickNode simply picks an element \( i \)
from the set \( Y \); details are given in Section 4. The
procedure CreateIneq(Y) generates the inequality, associated with the node-set \( Y \), that corresponds to the
half-space containing pos \( E \).

Because we also want to apply this procedure to networks where a single node could correspond to a
collapsed set of nodes in the original network (collapsing will be defined precisely later on), we introduce
the notation \( A(Y) \) to denote the set of all nodes in the
original network that corresponds to the set of nodes \( Y \)
in the collapsed network. Of course, if no collapsing
has taken place, then \( A(Y) = Y \). Remember that \( \beta(i) \)
is the total external flow at node \( i \), and \( \beta(i) > 0 \) means
net supply at node \( i \) and \( \beta(i) < 0 \) net demand.

```python
procedure CreateIneq(Y; set of nodes);
begin
  if \( A(Y) \neq \emptyset \) then
    create the inequality \( \sum_{i \in A(Y)} \beta(i) \leq 0 \)
  else
    create the inequality \( -\sum_{i \in \mathcal{N}} \beta(i) \leq 0 \);
end;
```

Note that if \( Y = \emptyset \), then \( \mathcal{N} \setminus Y = \mathcal{N} \), and hence there
must be no flow from nowhere to \( \mathcal{N} \), i.e. \( \mathcal{N} \) must not be
a net demander. Hence the interpretation of the
case \( A(Y) = \emptyset \) in the procedure is correct. Since this,
together with the inequality \( \sum_{i \in \mathcal{N}} \beta(i) \leq 0 \), implies that
\( \sum_{i \in \mathcal{N}} \beta(i) = 0 \), \( \beta(n) \) can be replaced by \( -\beta(1) - \ldots - \beta(n-1) \) in all the inequalities generated.

We shall now present a number of results to simplify
the procedure BasicFacets. The idea is to characterize
cuts \( Q \) that have \( Q^* = \emptyset \), since those, according
to Proposition 1, are the only interesting cuts.

**Proposition 2.** If \( Q^* = [Y, \mathcal{N} \setminus Y]^* = \emptyset \) then \( i \in Y \Rightarrow \)
\( F^*(i) \subseteq Y \).

**Proof.** Assume we have a cut \( Q \) with \( Q^* = \emptyset \) such that
\( i \in Y, j \notin Y \), but \( j \notin F^*(i) \). Then there is a directed
path from \( i \) to \( j \) such that at least one of the arcs in the
path is in \( Q^* \). Hence \( Q^* \neq \emptyset \), and the proposition is
true by contradiction. ■

Note that if \( [Y, \mathcal{N} \setminus Y]^* = \emptyset \) then \( Y = F^*(Y) \).
From Proposition 2 it follows:

**Proposition 3.** If \( j_1, j_2, \ldots, j_k \) is a set of arcs such that
\( j_k \sim (i_k, i_{k+1}) \) and \( i_1 = i_{k+1} \) then the nodes \( i_1, i_2, \ldots, i_k \) will
always be on the same side of a cut \( Q \) if \( Q^* = \emptyset \).

The following procedure looks for circuits satisfying
Proposition 3.

```
procedure CollapseNodes;
begin
  K := \mathcal{N} \setminus \emptyset;
  repeat
    PickNode(i, K);
    s := B^*(i) \cap F^*(i);
    if \( s \neq \emptyset \) then begin
      \( \mathcal{N} := (\mathcal{N} \setminus s) \cup \{i\}\);
      for all \( j \in \mathcal{N} \) do begin
        if \( s \subseteq B^*(j) \) then
          B^*(j) := \( (B^*(j) \setminus s) \cup \{i\}\);
        if \( s \subseteq F^*(j) \) then
          F^*(j) := \( (F^*(j) \setminus s) \cup \{i\}\);
      end;
    Collapse(s, i);
  end;
  K := K \setminus s;
  until K = \emptyset;
end;
```
The procedure Collapse is used to let one node \( i \) represent a set of nodes \( s \). It is given as follows.

```plaintext
procedure Collapse(Y: set of nodes; i: node);
begin
    A(i) := A(i) \cup A(Y);
    for all j \in Y \setminus \{i\} do
        A(j) := \emptyset;
end;
```

**Proposition 4.** If \( j \in B^*(i) \) and \( Y \) is some set of nodes with \( i, j \notin Y \) then \( \text{CreateIneq}(Y \cup \{i, j\}) \) and \( \text{CreateIneq}(Y \cup \{j\}) \) will create the same inequality.

**Proof.** \( F^*(i) \subseteq F^*(j) \).

Hence, when we consider cases where \( i \in Y \), we need not check any combinations with nodes \( j \in B^*(i) \).

Next follow some results showing that even if \( Q^* = \emptyset \) we do not necessarily obtain a facet of pos \( E \).

**Definition** An inequality is said to be dominated by a set of other inequalities if it can be written as a negative linear combination of the inequalities in the set.

Hence, a dominated inequality is redundant.

**Proposition 5.** If \( [Y, \mathcal{N} \setminus Y]^+ = \emptyset \) and \( Y = Y_1 \cup Y_2 \) with \( F^*(Y_1) \cap F^*(Y_2) = \emptyset \) then \( \text{CreateIneq}(Y) \) will not create a facet of pos \( E \).

**Proof.** The inequality created by \( \text{CreateIneq}(Y) \) will be dominated by those created by \( \text{CreateIneq}(Y_1) \) and \( \text{CreateIneq}(Y_2) \).

This is a special case of Theorem 2.1 of Prékopa and Boros[16] who also use the Gale-Hoffman characterization of feasibility to generate the faces of the polyhedron that bound the set of feasible flows for capacitated networks, see also Prékopa[15] and Prékopa and Boros.[17]

**Proposition 6.** If \( [Y, \mathcal{N} \setminus Y]^+ = \emptyset \) and \( \mathcal{N} \setminus Y = Z_1 \cup Z_2 \) with \( B^*(Z_1) \cap B^*(Z_2) = \emptyset \) then \( \text{CreateIneq}(Y) \) will not generate a facet of pos \( E \).

**Proof.** The inequality created by \( \text{CreateIneq}(Y) \) will be dominated by those created by \( \text{CreateIneq}(Y \cup Z_1) \), \( \text{CreateIneq}(Y \cup Z_2) \) and \( \text{CreateIneq}(\emptyset) \).

In other words, Propositions 5 and 6 say that whenever the subnetwork generated by \( Y \) or \( \mathcal{N} \setminus Y \) is not connected then \( \text{CreateIneq}(Y) \) does not create a facet of pos \( E \). We can also show the reverse result.

**Proposition 7.** If \( [Y, \mathcal{N} \setminus Y]^+ = \emptyset \) and there are no nonempty sets \( Y_1 \) and \( Y_2 \) such that \( Y = Y_1 \cup Y_2 \) with \( F^*(Y_1) \cap F^*(Y_2) = \emptyset \) and no nonempty sets \( Z_1 \) and \( Z_2 \) such that \( \mathcal{N} \setminus Y = Z_1 \cup Z_2 \) with \( B^*(Z_1) \cap B^*(Z_2) = \emptyset \), then \( \text{CreateIneq}(Y) \) will create a facet of pos \( E \).

**Proof.** To prove Proposition 7 we shall need some simple lemmas.

**Lemma 8.** Assume \( [Y, \mathcal{N} \setminus Y]^+ = \emptyset \), and assume there are no nonempty sets \( Z_1 \) and \( Z_2 \), such that \( \mathcal{N} \setminus Y = Z_1 \cup Z_2 \), with \( B^*(Z_1) \cap B^*(Z_2) = \emptyset \). Then if \( [K, \mathcal{N} \setminus K]^+ = \emptyset \) then \( \text{CreateIneq}(K) \) will create a facet of pos \( E \).

**Proof.** Since \( [K, \mathcal{N} \setminus K]^+ = \emptyset \) there are no arcs from \( Z_1 \) to \( Z_2 \). But there must be arcs from \( Z_1 \) to \( Z_1 \), such that \( B^*(Z_1) \cap B^*(Z_2) = \emptyset \), which we have assumed is not the case. But with arcs from \( Z_1 \) to \( Z_2 \), cannot satisfy Proposition 1. The set \( Y \cup Z_2 \) cannot satisfy Proposition 1 for exactly the same reasons.

**Lemma 9.** If \( [K_1, \mathcal{N} \setminus K_1]^+ = \emptyset \) and \( [K_2, \mathcal{N} \setminus K_2]^+ = \emptyset \), then \( \text{CreateIneq}(K_1 \cup K_2) \) will create a facet of pos \( E \).

**Proof.** There are no arcs going from \( K_1 \cup K_2 \) to \( \mathcal{N} \setminus (K_1 \cup K_2) \).

Let us now continue with the proof of Proposition 7. If \( Y \) does not create a facet, then \( \sum_{i \in \mathcal{N}} \beta(i) \leq 0 \) is dominated by cuts generated from sets \( Y_i \), \( i \in I_1, Z_i, i \in I_2 \) and sets \( K_i = (Y_i \cup Z_i), i \in I_3 \), where \( Y_i \subseteq Y \), \( Z_i \subseteq \mathcal{N} \setminus Y \), but such that \( Y_i = Y, i \in I_1 \) is not available.

In addition we have the cut \( \sum_{i \in \mathcal{N}} \beta(i) \leq 0 \) as explained when discussing Procedure CreateIneq.

If we are at all to use sets with their indices in \( I_2 \) and \( I_3 \), the corresponding inequalities must add up to

\[
\alpha \sum_{i \in \mathcal{N} \setminus Y} \beta(i) + \sum_{i \in Y} \lambda(i) \beta(i) \leq 0
\]

for some parameters \( \alpha \) and \( \lambda(i) \). Then the first part can be offset by \( -\alpha \sum_{i \in \mathcal{N} \setminus Y} \beta(i) \leq 0 \), leaving us with \( \sum_{i \in \mathcal{N}} (\lambda(i) - \alpha) \beta(i) \leq 0 \). If we end up with

\[
\gamma(i) \beta(i) + \sum_{i \in Y} \lambda(i) \beta(i) \leq 0
\]

for some \( \gamma(i) \)'s that are not the same, there is no way we can add a multiple of \( -\sum_{i \in \mathcal{N} \setminus Y} \beta(i) \leq 0 \) and cuts corresponding to sets with their indices in \( Y \), and end up with \( \sum_{i \in \mathcal{N}} \beta(i) \leq 0 \). Hence, if we use sets with their indices in \( I_2 \) or \( I_3 \), the corresponding cuts must add as explained.

But according to Lemmas 8 and 9, this can never happen. Whatever is missing in the sum will not be available because it will not be represented by a set satisfying Proposition 1. Hence, the only possibility is to use only sets with their indices in \( I_1 \).
Lemma 10. If \([Y, \mathcal{N} \setminus Y]^+ = \emptyset\), with \(Y = Y_1 \cup Y_2\), such that \(F^*(Y_1) \cap F^*(Y_2) = \emptyset\), then \([Y, \mathcal{N} \setminus Y_1]^+ = \emptyset\), we get \([Y_2, \mathcal{N} \setminus Y_2]^+ \neq \emptyset\).

Proof. The only way we could get \([Y_2, \mathcal{N} \setminus Y_2]^+ = \emptyset\) was to have no arcs from \(Y_2\) to \(Y_1\). But that contradicts the assumption that \(F^*(Y_1) \cap F^*(Y_2) = \emptyset\), since by assumption there are no arcs from \(Y_1\) to \(Y_2\).

Lemma 11. If \(Y_i \subseteq Y\) and \(Y_2 \subseteq Y\), with \([Y_i, \mathcal{N} \setminus Y_j]^+ = \emptyset\), \(i = 1, 2\), then also \([Y_1 \cup Y_2, \mathcal{N} \setminus (Y_1 \cup Y_2)]^+ = \emptyset\).

Proof. No arcs go from \((Y_1 \cup Y_2)\) to the rest of the network.

These two last lemmas show that there is no way we can add inequalities corresponding to cuts with their indices in \(I_i\), so as to end up with \(\sum_{i \in Y} \beta(i) \leqslant 0\). And this completes the proof of Proposition 7.

We can sum up the earlier results as follows.

**Theorem 12.** A cut \(Q = [Y, \mathcal{N} \setminus Y]\) with \(Q^+ = \emptyset\) represents a facet if and only if the network generated by the nodes in \(Y\) and the network generated by the nodes in \(\mathcal{N} \setminus Y\) are both connected.

Below follows a routine that checks if a subnetwork represented by the node-set \(W\) is connected.

---

**function** Connected\((W; \text{set of nodes}); \text{boolean};

**begin**

PickNode\((i, W)\);
Qlist := \([i]\);
Visited := \([i]\);

**while** Qlist \(\neq \emptyset\) **do begin**
PickNode\((i, Qlist)\);
Qlist := Qlist \(-\{i\}\);
s := \((B^*(i) \cup F^*(i)) \cap (W \setminus Visited)\);
Qlist := Qlist \cup s;
Visited := Visited \cup s;

end;
Connected := (Visited = W);

**end;**

Before we continue we shall define the set of supply nodes \(S\) and the set of demand nodes \(D\). Note that if \(\mathcal{N} = S \cup D\) we have a bipartite network.

**Definition.** Let \(S = \{i | B^*(i) = \{i\}\}\) and \(D = \{i | F^*(i) = \{i\}\}\).

Let us now present a number of propositions which are going to be useful when trying to apply Theorem 12. To make the reading a bit easier we show an example network in Figure 1. The network is minimal in the sense that it contains only arcs needed to describe...
feasibility. Even so, this is not a special case, except that it has no directed circuits. As outlined in Appendix A, using $F^*$ and $B^*$ will always result in a minimal network in this sense. The reader is encouraged to consult Appendix A at this point.

Although we shall apply the propositions to follow only on networks where all directed circuits have been collapsed, the results are applicable also to networks where directed circuits are present.

**Proposition 13.** If $k \in S$ and $j \in F^*(k) \setminus \{k\}$ such that $F^*(k) = \{k\} \cup F^*(j)$, then the nodes $j$ and $k$ can be collapsed into one node after the facet generated by $\text{CreateIneq}(\mathcal{N} \setminus \{i\})$ has been created.

**Proof.** (See Figure 1 with $k = 15$ and $j = 2$). First, note that with $Y = \mathcal{N} \setminus \{k\}$, $Q^* = \{1, 2, 3\} = \emptyset$. Second, the network generated by the (single) node in $\mathcal{N} \setminus Y$ is connected. Third, the network generated by the nodes in $Y$ is connected since removing arc $d$ does not disconnect the rest of the originally connected network because $F^*(k) = \{k\} \cup F^*(j)$. Hence $\text{CreateIneq}(\mathcal{N} \setminus \{k\})$ does create a facet.

Next we must show that for all other facets the nodes $j$ and $k$ will always be on the same side of the corresponding cut $Q$. Since $k \in Y \Rightarrow j \in Y$, the only possible way of splitting the nodes is to have $\{k\} \subset \mathcal{N} \setminus Y$ and $F^*(j) \subset Y$. (If $\mathcal{N} \setminus Y$ corresponds to the facet mentioned in the proposition.) But if $k \in \mathcal{N} \setminus Y$ together with some other node(s) in $\mathcal{N} \setminus F^*(j)$, the network generated by $\mathcal{N} \setminus Y$ must be disconnected, and hence the corresponding cut does not create a facet. (All connections between node $k$ and other nodes pass through nodes in $F^*(j)$.)

Similarly we have the following.

**Proposition 14.** If $k \in D$ and $j \in B^*(k) \setminus \{k\}$ such that $B^*(k) = \{k\} \cup B^*(j)$, then the nodes $j$ and $k$ can be collapsed into one node after the facet generated by $\text{CreateIneq}(\{i\})$ has been created.

**Proof.** (See Figure 1 with $k = 16$ and $j = 3$).

Looking at Figure 1 we realize that there are cases when it is better to look for node $j$ (in terms of the last two propositions), rather than node $k$. This is clarified below.

**Proposition 15.** If $j \in \mathcal{N} \setminus D$, $F^*(j) \setminus \{j\} \subseteq D$, and $B^*(i) = \{i\} \cup B^*(j)$ for all $i \in F^*(j) \setminus \{j\}$, then the nodes in $F^*(j)$ can be collapsed into one node after the facets generated by $\text{CreateIneq}(\{i\})$, for all $i \in F^*(j) \setminus \{j\}$, have been generated.

**Proof.** (Case A, see Figure 1 with $j = 6$.) Follows by recursion from Proposition 14.

**Proposition 16.** If $j \in \mathcal{N} \setminus S$, $B^*(j) \setminus \{j\} \subseteq S$, and $F^*(i) = \{i\} \cup F^*(j)$ for all $i \in B^*(j) \setminus \{j\}$, then the nodes in $B^*(j)$ can be collapsed into one node after the facets generated by $\text{CreateIneq}(\mathcal{N} \setminus \{i\})$, for all $i \in B^*(j) \setminus \{j\}$, have been generated.

**Proof.** (Case B, see Figure 1 with $j = 1$.)

**Proposition 17.** If $j \in D$, $k \in B^*(j) \setminus \{j\}$, $B^*(j) \setminus \{k\}$ is not connected, and $F^*(i) = \{i, j\}$ for all $i \in B^*(j) \setminus \{k\}$, then the nodes in $B^*(j) \setminus \{k\}$ can be collapsed into node $k$ after the facets generated by $\text{CreateIneq}(\mathcal{N} \setminus \{i\})$, for all $i \in B^*(j) \setminus \{k\}$ and $\text{CreateIneq}(B^*(j) \setminus \{k\})$ have been generated.

**Proof.** (Case C, see Figure 1 with $j = 9$.)

Note that if $B^*(j) \setminus \{k\} \subseteq \emptyset$, we are in the situation covered by Proposition 14.

**Proposition 18.** If $j \in S$, $k \in F^*(j) \setminus \{j\}$, $F^*(j) \setminus \{k\}$ is not connected, and $B^*(i) = \{i, j\}$ for all $i \in F^*(j) \setminus \{k\}$, then the nodes in $F^*(j) \setminus \{k\}$ can be collapsed into node $k$ after the facets generated by $\text{CreateIneq}(\{i\})$, for all $i \in F^*(j) \setminus \{k\}$ and $\text{CreateIneq}(F^*(j) \setminus \{k\})$ have been generated.

**Proof.** (Case D, see Figure 1 with $j = 12$.)

Again, if $F^*(j) \setminus \{k\} \subseteq \emptyset$ we are in the situation covered by Proposition 13. Corresponding to these results are the following four procedures.

```plaintext
procedure A(found : boolean); begin
K := \mathcal{N} \setminus D;
while K \neq \emptyset do begin
PickNode(j, K);
K := K \setminus \{j\};
t := F^*(j) \setminus \{j\};
if (t \subseteq D) and (B^*(i) = \{i\} \cup B^*(j) for all i \in t) then begin
for all i \in t do CreateIneq(\{i\});
D := (D \setminus F^*(j)) \cup \{j\};
\mathcal{N} := (\mathcal{N} \setminus F^*(j)) \cup \{j\};
Collapse(F^*(j), j);
for all i \in \mathcal{N} \setminus \{j\} do if F^*(j) \subseteq F^*(i) then
F^*(i) := (F^*(i) \setminus F^*(j)) \cup \{j\};
F^*(j) := \{j\};
found := true;
end;
end;
end;
```
procedure B(found: boolean);
begin
  K := \mathcal{N}\setminus S;
  while K \neq \emptyset do begin
    PickNode(j, K);
    K := K \setminus j;
    t := B^*(j) \setminus \{j\};
    if (t \subseteq S) and (F^*(i) = \{i\} \cup F^*(j) for all i \in t) then begin
      for all i \in t do Createlnq(\mathcal{N}\setminus \{i\});
      S := (S\setminus B^*(j)) \cup \{j\};
      \mathcal{N} := (\mathcal{N}\setminus B^*(j)) \cup \{j\};
      Collapse(B^*(j), j);
      for all i \in \mathcal{N}\setminus \{j\} do if B^*(j) \subseteq B^*(i) then
        B^*(i) := (B^*(i)\setminus B^*(j)) \cup \{j\};
      B^*(j) := \{j\};
      found := true;
    end;
  end;
end;

procedure D(found: boolean);
begin
  K := S;
  repeat
    PickNode(j, K);
    K := K \setminus j;
    L := F^*(j) \setminus \{j\};
    while L \neq \emptyset do begin
      PickNode(k, L);
      L := L \setminus k;
      t := F^*(j) \setminus (F^*(k) \cup \{j\});
      if (t \subseteq \emptyset) and (B^*(i) = \{i, j\} for all i \in t) then begin
        for all i \in t do Createlnq(\mathcal{N}\{i\});
        Createlnq(\mathcal{N}\{t \cup \{j\}\});
        Collapse(t \cup \{j\}, k);
        \mathcal{N} := \mathcal{N}\{t \cup \{j\}\};
        for all i \in \mathcal{N} do B^*(i) := B^*(i) \setminus \{j\};
        found := true;
      end;
    end;
  until K = \emptyset;
end;

procedure C(found: boolean);
begin
  K := D;
  repeat
    PickNode(j, K);
    K := K \setminus j;
    L := B^*(j) \setminus \{j\};
    while L \neq \emptyset do begin
      PickNode(k, L);
      L := L \setminus k;
      t := B^*(j) \setminus (B^*(k) \cup \{j\});
      if (t \subseteq S) and (F^*(i) = \{i, j\} for all i \in t) then begin
        for all i \in t do Createlnq(\mathcal{N}\{i\});
        Createlnq(\mathcal{N}\{t \cup \{j\}\});
        Collapse(t \cup \{j\}, k);
        \mathcal{N} := \mathcal{N}\{t \cup \{j\}\};
        for all i \in \mathcal{N} do F^*(i) := F^*(i) \setminus \{j\};
        found := true;
      end;
    end;
  until K = \emptyset;
end;

These routines can be used as follows in Procedure TreeRemoval.

procedure TreeRemoval;
begin
  repeat
    found := false;
    A(found), B(found), C(found), D(found);
  until not found;
end;

Procedure TreeRemoval will be the first part of our facet-generating procedure. All nodes that survive Procedure TreeRemoval are on circuits or on paths joining circuits. There are no trees attached to the circuits. Procedure TreeRemoval generates only facets, no lower-dimensional faces. Below we define a special set of faces. We shall apply the definition and the results that follow to the network obtained after applying Procedure TreeRemoval. However, the validity of the results does not require such preprocessing.
Definition. The following set of inequalities are defined to be the basic set of faces:

$$\beta(i) \leq 0 \quad \text{for all } i \in D$$

$$\sum_{j \neq i} \beta(j) \leq 0 \quad (\iff \beta(i) \geq 0) \quad \text{for all } i \in S$$

$$\sum \beta(i) \leq 0$$

$$-\sum \beta(i) \leq 0$$

Some of the faces in the basic set of faces will be facets, however, others will not, consult Propositions 5 and 6.

Proposition 19. If $[Y \setminus \mathcal{N} \setminus Y]^* = \emptyset$ and $\mathcal{N} \setminus Y \subseteq S$, then the inequality created by $\text{CreateIneq}(Y)$ will be dominated by the basic set of faces.

Proof. Since $\beta(i) \geq 0$ for all $i \in \mathcal{N} \setminus Y$ and $\sum \beta(i) = 0$, we have that $\sum_{i \in Y} \beta(i) \leq 0$. ■

From this we obtain:

Proposition 20. If $[Y \setminus \mathcal{N} \setminus Y]^* = \emptyset$ and $\mathcal{N} \setminus Y \subseteq S$, then if $[Z, \mathcal{N} \setminus Z]^* = \emptyset$ and $Y \subseteq Z$ we have that the inequality created by $\text{CreateIneq}(Z)$ is dominated by the basic set of faces.

Hence, when using BasicFacets, we find a set $W$ satisfying Proposition 19, we need not continue to add elements to $W$ as all sets created thereafter (along that branch of the enumeration tree) will satisfy Proposition 20. The following two propositions are the mirror images of these results.

Proposition 21. If $[Y \setminus \mathcal{N} \setminus Y]^* = \emptyset$ and $Y \subseteq D$, the inequality created by $\text{CreateIneq}(Y)$ will be dominated by the basic set of faces.

Proposition 22. If $[Y \setminus \mathcal{N} \setminus Y]^* = \emptyset$ and $Y \subseteq D$, then if $[Z, \mathcal{N} \setminus Z]^* = \emptyset$ and $Z \subseteq Y$ we have that the inequality created by $\text{CreateIneq}(Z)$ will be dominated by the basic set of faces.

Since, in the procedure BasicFacets, we are adding elements to $W$, this last proposition is not very useful. Clearly, there is a dual version of BasicFacets where Proposition 22 is useful, but Proposition 20 is not.

We are now ready to present a much more efficient version of BasicFacets.

```
procedure AllFacets;
begin
  CollapseNodes;
  TreeRemoval;
  CreateIneq($\mathcal{N}$);
  CreateIneq($\emptyset$);
  for all $i \in S$ do if $\mathcal{N} \setminus \{i\} \neq \emptyset$ then
    if Connected($\mathcal{N} \setminus \{i\}$) then
      CreateIneq($\mathcal{N} \setminus \{i\}$);
  for all $i \in D$ do if $\mathcal{N} \setminus \{i\} \neq \emptyset$ then
    if Connected($\mathcal{N} \setminus \{i\}$) then
      CreateIneq($\{i\}$);
  $Y := \mathcal{N}$;
  $W := \emptyset$;
  Facets($Y, W$);
end;
```

```
procedure Facets($Y, W$): set of nodes;
begin
  if $Y \neq \emptyset$ then begin
    PickNode($i, Y$);
    $Y := Y \setminus \{i\}$;
    Facets($Y, W$);
    $W := W \cup F^*(i)$; (* Proposition 2 *)
    $Y := Y \setminus (F^*(i) \cup B^*(i))$; (* Propositions 2 & 4 *)
    if $\mathcal{N} \setminus W \not\subseteq S$ then begin (* Propositions 19 & 20 *)
      if Connected($W$) and Connected($\mathcal{N} \setminus W$) (* Prop. 5 & 6 *)
      and $W \not\subseteq D$ then (* Proposition 21 *)
        CreateIneq($W$);
    end;
  end;
end;
```

Again $\beta(n)$ should be replaced by $-\beta(1) - \ldots - \beta(n-1)$. Clearly, the number of facets generated by AllFacets is still, in the worst case, exponential in the number of nodes. However, it must be remembered that we start by collapsing all directed circuits, and that the use of Propositions 2 to 22 will reduce the number of sets that need to be checked, particularly if the network density is high. We shall in a later section show how some additional preprocessing can help reduce the network even further.

In terms of the stochastic program in the introduction, we shall use each facet to generate inequalities in terms of $x$ that will be added to the constraints $Ax = a$. 
We are then guaranteed that all first stage solutions generated by the L-shaped procedure are in fact feasible. We are left with a problem with relatively complete recourse. More specifically, since the total external flow \( \beta(i) = \sum_{j} H_{i}(i,j)b(j) + \sum_{k} H_{2}(i,k)x(k) \), we can do the following. From the facet 

\[
\sum_{i \in W} \beta(i) = \sum_{i \in W} \left( \sum_{j} H_{1}(i,j)b(j) + \sum_{k} H_{2}(i,k)x(k) \right) = 0
\]

we obtain the following inequality in terms of \( x \) (recall \( \beta(n) \) is replaced by \( -\beta(1) - \ldots - \beta(n-1) \), and that the above must be true for all \( \beta \in (\text{supp}b) \)):

\[
\sum_{i \in W} \sum_{k} H_{2}(i,k)x(k) \leq \sum_{j} \min_{i \in W} \left( \sum_{j} H_{1}(i,j) \right)b(j).
\]

Since several pos \( E \) facets can generate the same inequalities, the inequalities above should be subjected to a frame finding procedure, such as for example Wets and Witzgall,\(^{27}\) together with the rows of \( A \) (rewritten as inequalities). That could substantially reduce the number of inequalities needed to get relatively complete recourse.

### 2. Numeric Results for Facet Generation

The procedures of the previous section have been implemented in Pascal on a Prime 9950. By using Pascal we were able to implement the set relations directly. Our implementation does not allow for more than 256 nodes. However, it is simple to expand this. (A set in Pascal has a maximum of 256 members.) For more details see Wallace.\(^{23}\)

All example networks in this section have been generated by the code NETGEN by Klingman et al.\(^{14}\) The necessary input parameters for the examples (except the number of arcs and nodes) can be found in the Appendix B of this paper. The number of arcs and nodes can be found in Table I.

The examples can be divided into two parts, namely Examples A–M, which are dense networks, and Examples N–U, which are sparse networks. The first set illustrates the potential of the routine CollapseNodes, whereas the second is more useful for testing AllFacets. Several of the examples in the second set have circuits in them (not directed circuits) after the call to CollapseNodes.

Note that from the results in the previous section, we can easily say that if a network is a tree after the call to CollapseNodes, then the number of facets is one higher than the number of nodes left, since one facet is added each time a node is removed, until the last node generates the two facets \( \sum \beta(i) \leq 0 \) and \( -\sum \beta(i) \leq 0 \). However, some of the examples have circuits in them and we may still end up with the number of facets equal to the number of nodes plus 1.

### Table I

<table>
<thead>
<tr>
<th>Ex. Nodes</th>
<th>Arcs</th>
<th>CPU I</th>
<th>CPU II</th>
<th>CPU III</th>
<th>CPU IV</th>
<th>Facets</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 40</td>
<td>390</td>
<td>0.26</td>
<td>0.00</td>
<td>1</td>
<td>0.02</td>
<td>0.01</td>
</tr>
<tr>
<td>B 40</td>
<td>390</td>
<td>0.28</td>
<td>0.00</td>
<td>1</td>
<td>0.01</td>
<td>0.02</td>
</tr>
<tr>
<td>C 40</td>
<td>390</td>
<td>0.27</td>
<td>0.00</td>
<td>1</td>
<td>0.01</td>
<td>0.02</td>
</tr>
<tr>
<td>D 40</td>
<td>390</td>
<td>0.23</td>
<td>0.07</td>
<td>11</td>
<td>0.05</td>
<td>0.02</td>
</tr>
<tr>
<td>E 40</td>
<td>390</td>
<td>0.23</td>
<td>0.07</td>
<td>11</td>
<td>0.04</td>
<td>0.03</td>
</tr>
<tr>
<td>F 40</td>
<td>390</td>
<td>0.23</td>
<td>0.08</td>
<td>11</td>
<td>0.04</td>
<td>0.03</td>
</tr>
<tr>
<td>G 40</td>
<td>195</td>
<td>0.20</td>
<td>0.07</td>
<td>11</td>
<td>0.04</td>
<td>0.03</td>
</tr>
<tr>
<td>H 40</td>
<td>195</td>
<td>0.22</td>
<td>0.07</td>
<td>12</td>
<td>0.05</td>
<td>0.03</td>
</tr>
<tr>
<td>I 40</td>
<td>195</td>
<td>0.21</td>
<td>0.07</td>
<td>11</td>
<td>0.05</td>
<td>0.02</td>
</tr>
<tr>
<td>J 40</td>
<td>2200</td>
<td>0.51</td>
<td>0.19</td>
<td>26</td>
<td>0.12</td>
<td>0.05</td>
</tr>
<tr>
<td>K 100</td>
<td>4000</td>
<td>1.49</td>
<td>0.32</td>
<td>26</td>
<td>0.12</td>
<td>0.04</td>
</tr>
<tr>
<td>L 200</td>
<td>4000</td>
<td>6.08</td>
<td>0.97</td>
<td>31</td>
<td>0.14</td>
<td>0.06</td>
</tr>
<tr>
<td>M 200</td>
<td>8000</td>
<td>5.90</td>
<td>1.49</td>
<td>51</td>
<td>0.30</td>
<td>0.10</td>
</tr>
<tr>
<td>N 20</td>
<td>50</td>
<td>0.07</td>
<td>0.00</td>
<td>1</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>O 30</td>
<td>50</td>
<td>0.11</td>
<td>0.01</td>
<td>4</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>P 20</td>
<td>50</td>
<td>0.05</td>
<td>0.03</td>
<td>16</td>
<td>0.13</td>
<td>0.11</td>
</tr>
<tr>
<td>Q 30</td>
<td>50</td>
<td>0.11</td>
<td>0.07</td>
<td>18</td>
<td>0.16</td>
<td>0.10</td>
</tr>
<tr>
<td>R 50</td>
<td>100</td>
<td>0.28</td>
<td>0.19</td>
<td>36</td>
<td>0.42</td>
<td>1.92</td>
</tr>
<tr>
<td>S 50</td>
<td>100</td>
<td>0.28</td>
<td>0.15</td>
<td>36</td>
<td>0.35</td>
<td>6.24</td>
</tr>
<tr>
<td>T 100</td>
<td>495</td>
<td>1.24</td>
<td>0.64</td>
<td>43</td>
<td>0.27</td>
<td>0.11</td>
</tr>
<tr>
<td>U 100</td>
<td>495</td>
<td>1.28</td>
<td>0.62</td>
<td>42</td>
<td>0.25</td>
<td>0.08</td>
</tr>
</tbody>
</table>

*All CPU times are in seconds and exclude I/O time. CPU I = time to find \( F^* \) and \( B^* \), CPU II = time for CollapseNodes, CPU III = time for TreeRemoval, CPU IV = time for Facets. \( n\)-red is the number of nodes left after CollapseNodes.*

Let us point out the important difference between the facets generated by the Procedure TreeRemoval and the Procedure Facets. Whenever a facet is generated in TreeRemoval, a node is removed. In the Procedure Facets, it is the Function Connected that helps us generate exclusively facets. However, Connected does not prevent us from visiting all these cuts which do not generate facets, it only prevents us from generating the facet. Therefore, although we would obtain the same result if Facets was called without a previous call to TreeRemoval, the CPU time would increase enormously. Also the number of cuts that were considered would grow. For some of the examples in Table I this number runs in the hundreds of thousands. Also, if Facets is used without Connected, the number of faces generated is of the same magnitude, namely several hundred thousands. Hence, without TreeRemoval and Connected, we would not be able to generate the facets for any reasonable size networks.

Note that for the large, dense networks, most of the time is spent calculating \( F^* \) and \( B^* \). This is not very surprising as the complexity is \( \sigma(n^3) \). For the more complicated networks, in terms of facets, we see that most time is spent in Facets. Or more precisely, if the number of nodes left after the call to TreeRemoval is large, the network has a complicated structure, and the
execution time is going to be high. The reason is that the computational complexity of Facets is exponential in the number of remaining nodes. From this we can also observe that the problem of finding all facets of pos $E$ is exponential in the number of nodes only if the network contains undirected cycles, otherwise it is polynomial.

As can be seen from Table I, for larger networks, a lot of time is spend on finding transitive closures. To partly get around that, an alternative approach would be to start out by looking for strongly connected components (complexity $O(m)$), then collapse each such component into one node (as in CollapseNodes), and only thereafter apply CreateFstar.

If the number of strongly connected components is $n_1$, and $n_1$ is very close to $n$, this is likely to increase CPU-times, whereas if $n_1$ is much smaller than $n$, we are likely to decrease CPU-times. In both cases, the overall complexity is $O(n^3)$. There is no way we can escape that.

In Figure 1 we presented an example network. The cone pos $E$ corresponding to that network has 18 facets. These are all given in Table II.

Procedure PickNode is used in Facets. Its task is to pick the next node in $Y$ to be added to $W$. We have tried to do that in a number of ways. (NETGEN numbers its nodes such that the supply nodes come first and the demand nodes last.)

- Pick the node $i$ in $Y$ with the highest number.
- Pick the node $i$ in $Y$ with the lowest number.
- Pick the node $i$ in $Y$ that has most nodes in $(F^\star(i) \cup B^\star(i)) \cap Y$.

The first choice is motivated by the fact that if $\mathcal{N} \setminus W \subseteq S$ then a lot of work is avoided in Facets, line 10. Also, if $W \subseteq D$ some work is avoided in line 14. Hence, we try to make $W$ a subset of $S$ as quickly as possible. The second choice is motivated by the idea that nodes in $S$ have most nodes in their sets $F^\star$ and that can be useful for Facets in line 7. The third choice is motivated by the 8th line in Facets where we subtract $F^\star \cup B^\star$ from $Y$. The more we can subtract, the fewer combinations we need to check.

All our tests have shown that the first choice is best. It is better than the second choice because its effect discussed in the previous paragraph is more important. (The cost is the same for these two choices.) The third choice is extremely bad because of the cost involved in finding the best node. That is a cost one is not willing to pay. All results presented in this paper use the first choice.

3. Simplifying the Check for Feasibility

When calling the procedure CheckFeas (see Appendix C), a set $FSET$ is needed. This is a set of realizations of $b$ such that if $Q(x, b, d)$ is feasible for all $b \in FSET$, then $Q(x, b, d)$ is feasible for all $b \in (supp b)$ for a fixed $x$. Note that $d$ is of no importance for feasibility. Hence, below, we shall not refer to $d$. If nothing is known about the problem, one might have to let $FSET$ contain all the extreme points in $(supp b)$. Hence, an exponential number of values will have to be checked. Checking this is computationally infeasible when the total number of random nodes increases above about 10 or 15. On the other hand, if pos $E = \mathcal{N}^{n-1}$, $FSET = \emptyset$, since any right hand side will make $Q(x, b, d)$ feasible. This is called complete recourse. If it is known that $Q(x, b, d)$ is feasible for all $b$ if $x \in [x \mid Ax = a, x \geq 0]$ we have relatively complete recourse, and again $FSET = \emptyset$. This will always be the case if the facet generation technique of the previous sections has been used.

From the calculus for convex cones (for details consult the proof of Proposition 5.15 of Wets[25]) we know that if $C$ is a convex cone contained in pos $E$ and $Q(x, b, d) < \infty$, i.e. the problem that defines $Q(x, b, d)$ is feasible. Then $Q(x, b, d) < \infty$ whenever $H_x(b - \hat{b}) \in C$.

The same arguments (with $C$ a ray) yield the following result that can be used directly.

**Proposition 23.** Fix $x$, and let $e = (0, \ldots, 0, 1, 0, \ldots, 0)$ be the unit identity vector with 1 in the $i$th position. If $Q(x, b_i, d) < \infty$ and

$$Ey = H_x e, \quad y \geq 0$$

is feasible, then $Q(x, b_2, d) < \infty$, for all $b_1$ and $b_2$ where $b_1(i) < b_2(i)$ and $b_1(j) = b_2(j)$ for $i \neq j$. Furthermore, if $Q(x, b_i, d) < \infty$ and

$$Ey = -H_x e, \quad y \geq 0$$

is feasible then $Q(x, b_2, d) < \infty$ for all $b_1$ and $b_2$ where $b_1(i) > b_2(i)$ and $b_1(j) = b_2(j)$ for $i \neq j$.

Hence, if $Ey = H_x e, y \geq 0$ is feasible, $b(i)$ can be fixed at its minimal value for all $b \in FSET$. Clearly, if $Ey = -H_x e$, is feasible, $b(i)$ can be fixed at its maximal value. Hence, for each node where $b(i)$ can be fixed at

---

**TABLE II**

Facets for the Example in Figure 1, Represented by $Z$ or \( \mathcal{N} \setminus Z \) (Whichever Is Simpler), for the Cases Where the Cut $Q = [Z, \mathcal{N} \setminus Z]$ Has $Q^\star = \emptyset$ and Satisfies the Necessary Connectivity Requirements

| $Z$ | \{7, 8, 13, 14, 16\} | \{9, 10, 11\} | \{6, 7, 8\} | \{2, 6, 7, 8, 9, 10, 11, 15\} | \{3, 6, 7, 8, 12, 13, 14, 16\} | $\emptyset$, $\mathcal{N}$ |
| \( \mathcal{N} \setminus Z \) | \{4, 5\} | \{10\} | \{11\} | \{15\} | \{12, 13, 14\} | \{1, 4, 5\} |
its upper or lower bound, the number of realizations in \( FSET \) will be divided by two.

The previous proposition applies to any uncapacitated linear program. Let us now turn to special results for uncapacitated networks.

For the case when each random variable affects only one node in the network we have the following two results. Without loss of generality assume that all entries in \( H_i \) are positive, and let \( r(i) \) be the row containing a nonzero in column \( i \). (Clearly similar results exist for a negative entry.)

**Proposition 24.** Assume each column of \( H_i \) has at most one non-zero entry. Then the problem \( Ey = H_i e_i \), \( y \geq 0 \) is feasible if there is a directed path from node \( r(i) \) to the slack node \( n \).

**Proof.** Obvious. ■

**Theorem 25.** Assume each column of \( H_i \) has at most one non-zero entry. Let \( b^* \) be a random vector with support \((\text{supp}\, b^*)\). More precisely, let \((\text{supp}\, b^*)(i) = [b(i)_{\min}, b(i)_{\max}] \) if \( r(i) \in E^*(n) \backslash \{n\} \) and \((\text{supp}\, b^*)(i) = [b(i)_{\min}, b(i)_{\max}] \) if \( r(i) \in E^*(n) \backslash \{n\} \). For all other \( i \), let \((\text{supp}\, b^*)(i) = [b(i)_{\min}, b(i)_{max}] \). Then for a fixed \( x \), if \( Q(x, b^*, d) < \infty \) for all \( b^* \in (\text{supp}\, b^*) \), then \( Q(x, b, d) < \infty \) for all \( b \in (\text{supp}\, b) \).

As a special result we obtain the following theorem.

**Theorem 26.** Assume each column of \( H_i \) has at most one non-zero entry. If, for each node \( i \) in a network corresponding to a non-zero row of \( H_i \), there either exists a directed path from node \( i \) to the slack node or a directed path from the slack node to node \( i \), then there exists a \( b \) such that if, for a fixed \( x \), \( Q(x, b, d) < \infty \) then \( Q(x, b, d) < \infty \) for all \( b \in (\text{supp}\, b) \).

Consider the example in Figure 2. Assume that node \( n = 6 \) is the slack node. Let \( b = (b_1, \ldots, b_5) \) with \((\text{supp}\, b) = [-1, 1]^5 \). Since \( E^*(6) = \{1, 3, 4, 5\} \) and \( B^*(6) = \{5, 6\} \) we get that \((\text{supp}\, b^*)\) (see Theorem 25) is as follows

\[
(\text{supp}\, b^*(i)) = \{1\} \text{ for } i \in \{1, 3, 4\} \\
(\text{supp}\, b^*(5)) = \{-1\} \text{ and} \\
(\text{supp}\, b^*(2)) = \{-1, 1\}
\]

Hence, to check feasibility, we must check only the two realizations \((1, -1, 1, 1, -1)\) and \((1, 1, 1, 1, -1)\). Of course, these results are independent of the first-stage decisions \( x \).

**4. Network Reductions**

So far we have been concerned with preprocessing that would affect feasibility checking. This section is mostly concerned with those operations that could speed up the calculation of \( \mathcal{E}(x) = E[Q(x, b, d)] \). The goal is to decrease the size of the network, and to do so we rely on two very simple (if not obvious) reductions.

Let as before \( \beta(i) \) be the external flow at node \( i \). The first reduction is to remove all nodes in the node set \( V = \{i | \beta(i) = 0\} \), called the set of transshipment nodes. The second reduction removes all arcs that prevent a node in

\[
S^* = \{i | \Pr[\beta(i) > 0] > 0, \Pr[\beta(i) < 0] = 0\}
\]

from being in \( S \) and all arcs that prevent a node in

\[
D^* = \{i | \Pr[\beta(i) < 0] > 0, \Pr[\beta(i) > 0] = 0\}
\]

from being in \( D \). The second reduction also affects feasibility as we shall see later. The probabilities in the definitions of \( V, S^* \) and \( D^* \) refer to variations in both \( x \) and \( b \). This might seem restrictive. However, in applications one will expect both \( H_1 \) and \( H_2 \) to have many zero rows, and the nonzero rows will typically not be the same rows for \( H_1 \) and \( H_2 \). For the example we mentioned in the Introduction, \( H_1 \) affects only the supply nodes (fishing quotas) and \( H_2 \) the demand nodes (plant capacities). In that example there are no transshipment nodes. (Or alternatively, the transshipment nodes have already been removed.) Also note that in the example there is never a question of the sign of \( \beta(i) \).

If one wishes to use the facet generating procedures presented earlier in this paper, one should always remove transshipment nodes first. It is not so clear whether or not the arcs should be removed first. It is good for the facet generating procedure to have many nodes in \( S \) and, to some extent, also in \( D \), but at the same time the arc removal will decrease the number of elements in \( E^* \) and \( B^* \), and that is not so good news. It is an empirical question, which we will discuss in the next section, as to which approach might be better.

The arc removal can also be useful if one wishes to solve the overall two-stage stochastic optimization problem by for example "trickling down" as presented
in Wets\cite{26} and Wallace\cite{21} since it might reduce the number of possible dual feasible bases. However, the result can also be an increased number of arcs and thereby dual feasible bases.

We shall in this section need to talk about arcs in series and arcs in parallel. Two arcs \(i \) and \(j \) are said to be in series if \( T(i) = O(j) \), \( F(O(j)) = \{j\} \) and \( B(T(i)) = \{i\} \). Two arcs are in parallel if they start and end in the same node.

The main idea behind node removal is as follows. Assume node \(k\) is to be removed. A new arc is then added between the originating node of every arc in \( B(k) \) and the terminating node of every arc in \( F(k) \). The cost of the new arc equals the sum of the costs of the two arcs it replaces. This is a meaningful term also if one or more of the arcs are random.

We shall call this process series reduction, even if the two arcs are not in series as defined above. After all necessary series reductions have been performed, node \(k\) is removed. If the new arcs have created arcs in parallel, these are replaced by a new arc whose cost equals the minimum of the costs of the two arcs that were in parallel. This is called parallel reduction. Details are given below.

There are three cases when the node removal does not change the expected minimum cost of the problem at hand. Those are

1. There is only one arc in \( F(k) \), and it is non-random.
2. There is only one arc in \( B(k) \), and it is non-random.
3. All arcs in \( F(k) \cup B(k) \) are non-random.

In all other cases the result will be a new network with an expected minimal cost that bounds the original expected minimal cost from below. The reason is that we are disregarding path dependencies. See also the work by Kamburovski\cite{13} and Dodin.\cite{71}

The arc removal is very similar. Assume \(k \in S^*\). One then adds new arcs as if one were to remove node \(k\) as described above. But instead of removing node \(k\) and all arcs incident to it, one only removes arcs in \(B(k)\), and keeps \(k\) and the arcs in \(F(k)\). That way node \(k\) qualifies for the set \(S\).

The reason why removing arcs entering a node in \(S^*\) is allowed, is that such a node will never receive flow from other nodes except for the purpose of sending it along to some other node. One then replaces this indirect route by a direct arc. The arc removal is exact in the same situations as those making the node removal exact.

Below follows the main procedure. We have set it up so that one can choose between a network that generates an upper bound on the expected minimal cost of the original network, and a network that yields a lower bound. These two networks will have the same arcs and nodes, but different costs associated with the arcs. Clearly one can devise a procedure that finds both bounds at the same time.

```
procedure Reduce(mode: boundtype);
begin
  repeat
    if arcs \(k\) and \(k'\) are in parallel then
      ParallelRemove\((k, k')\);
  until no more arcs in parallel;
  repeat
    PickNode\((i, V)\);
    if mode = upperbound then MakeDet\((i)\);
    NodeRemoval\((i)\);
    until \(V = \emptyset\);
    \(S^* := S^* \setminus S\);
    \(D^* := D^* \setminus D\);
  repeat
    PickNode\((k, S^*)\);
    if mode = upperbound then MakeDet\((k)\);
    ArcRemoval\((S^*)\);
    \(S := S \cup \{k\}\);
    \(S^* := S^* \setminus \{k\}\);
  until \(S^* = \emptyset\);
  repeat
    PickNode\((k, D^*)\);
    if mode = upperbound then MakeDet\((k)\);
    ArcRemoval\((D^*)\);
    \(D := D \cup \{k\}\);
    \(D^* := D^* \setminus \{k\}\);
  until \(D^* = \emptyset\);
end;
```

We shall explain the procedure ParallelRemove below. However, its effect is to make sure that there are never two arcs between the same pair of nodes. The use of ParallelRemove does not change the solution of the original problem.

```
procedure PickNode\((i: \text{node}; V: \text{set of nodes})\);
The purpose of this procedure is to pick a node \(i \) in \(V\) as long as \(V\) is not empty. For most
always the choice is immaterial. For calls from
Reduce, when random arcs are present, we
suggest the following choice.
begin
  if there is a node \(j \in V\) with \(F(j)\) or \(B(j)\)
    singleton sets then
    \(i := j\);
  else
    if \(j \in V\) has the lowest number of random arcs
      in \(F(j) \cup B(j)\) then
      \(i := j\);
end;
```
Other criteria are possible, such as picking the node with the lowest number of arcs (not necessarily random) entering or leaving. However, for calls from Reduce, when random arcs are present, one should always start with nodes $j$ where $F(j)$ or $B(j)$ are singleton sets.

```
procedure MakeDet($i$ : node);
The purpose of this procedure is to replace the distributions of the costs of some arcs entering or leaving node $i$ by their expectations.
begin
  if $F(i) = \{k\}$ then
    replace $d(k')$ by its mean
  else if $B(i) = \{k'\}$ then
    replace $d(k')$ by its mean
  else for all $k \in F(i) \cup B(i)$ do
    replace $d(k)$ by its mean;
end;
```

Note that it is Procedure MakeDet that yields us the difference between upper and lower bounding. When one or more of the random costs $d(k)$ are replaced by their expectations, it follows from Jensen's inequality that the result is an upper bound. At the same time, we have made sure that the node or arc removal that follows will be exact, so that we do in fact have an upper bound.

Next follows the main procedure for node removal. There will be no arcs in parallel when the procedure starts and no arcs in parallel when it finishes.

```
procedure NodeRemoval($k$ : node);
begin
  for all $j \in B(k)$ do
    for all $i \in F(k)$ do begin
      SeriesReduction($j, i, t$);
      $F(O(j)) := F(O(j)) \cup \{t\}$;
      $B(T(i)) := B(T(i)) \cup \{t\}$;
      if $t' \sim (O(t), T(i))$ then
        ParallelRemove($t, t'$);
    end;
  for all $j \in B(k)$ do $F(O(j)) = F(O(j)) \setminus \{j\}$;
  for all $i \in F(k)$ do $B(T(i)) = B(T(i)) \setminus \{i\}$;
  $\mathcal{N} := \mathcal{N} \setminus \{k\}$;
end;
```

As explained above, the idea is to perform a series reduction on every pair of arcs entering and leaving node $k$. If that creates arcs in parallel, they are replaced by one new arc representing them both. Note that series reductions can be performed on arcs even if they are not in series as defined above. Let us now present the procedure SeriesReduction.

```
procedure SeriesReduction($i, j, k$ : arcs);
begin
  $(\supp(k)) := (\supp(i)) \times (\supp(j))$;
  for all $s \in (\supp(k))$ do
    $\Pr[d(k) = s] := \sum \Pr[d(i) = s - \tau] \Pr[d(j) = \tau]$;
end;
```

SeriesReduction does not remove the original arcs. ParallelReduction is similar to SeriesReduction, except that it looks at arcs in parallel.

```
procedure ParallelReduction($i, j, k$ : arcs);
begin
  $(\supp(k)) := (\supp(i)) \cup (\supp(j))$;
  for all $s \in (\supp(k))$ do
    $\Pr[d(k) \leq s] := \Pr[d(i) \leq s] + \Pr[d(j) \leq s]$
    $- \Pr[d(i) \leq s] \Pr[d(j) \leq s]$;
end;
```

Next follows the procedure that removes the two arcs in parallel.

```
procedure ParallelRemove($i, j$ : arcs);
begin
  ParallelReduction($i, j, k$);
  $F(O(i)) := F(O(i)) \setminus \{k\}$;
  $B(T(i)) := B(T(i)) \setminus \{k\}$;
end;
```

This completes the procedures for node removal. Let us now turn to the procedures for removing arcs.
procedure ArcRemovalS(k': node);
begin
  for all i ∈ B(k') do begin
    k := O(i);
    for all j ∈ F(k') do begin
      SeriesReduction(i, j, t);
      if i' ~ (k, T(j)) then
        ParallelRemove(t', t)
      else begin
        F(k) := F(k) ∪ {t};
        B(T(j)) := B(T(j)) ∪ {t};
      end;
    end;
  end;
  for all i ∈ B(k') do begin
    F(O(i)) := F(O(i)) \ {i};
    B(k') := Ø;
  end;
end;

procedure ArcRemovalD(k': node);
begin
  for all i ∈ F(k') do begin
    k := T(i);
    for all j ∈ B(k') do begin
      SeriesReduction(j, i, t);
      if i' ~ (O(j), k) then
        ParallelRemove(t', t)
      else begin
        F(O(j)) := F(O(j)) ∪ {t};
        B(k) := B(k) ∪ {t};
      end;
    end;
  end;
  for all i ∈ F(k') do begin
    B(T(i)) := B(T(i)) \ {i};
    F(k') := Ø;
  end;
end;

There is a potential problem when applying ArcRemovalS and ArcRemovalD as described above. The problem occurs if a node k ∈ S* is also in a directed circuit. By putting ArcRemovalS before CollapseNodes, the node k will be removed from the circuit, thereby making the circuit smaller. Hence, the resulting network will be larger than otherwise necessary. We therefore suggest that ArcRemovalS is applied after CollapseNodes, and only if k ∈ K and A(k) = {k}, i.e., only if k has not been part of a node collapse. In the next section follows results concerning the use of ArcRemovalS and ArcRemovalD after CollapseNodes.

When these reductions have been performed, a large scale network might be reduced to a very small problem. Therefore, even algorithmic procedures that have been abandoned because they can only be used for toy problems might in fact be useful in real world examples. Also note that the reduced network will most probably be very dense.

The savings that might result from such reductions are not so easy to evaluate. However, Bertsekas and Tseng describe a network code from the public domain called RELAXT-II (which most probably is the most efficient code available today), and indicate that a substantial decrease in the number of nodes in a network can yield savings in CPU-times of several orders of magnitude.

5. Numeric Results for Arc Removal

The previous section was primarily concerned with simplifying the calculation of F(x) by reducing the size of the network recourse problem. However, the reductions affect also the facet generation as was described in an earlier section. The reason is that we by introducing new information reduce the need for inequalities describing feasibility. Node removal will always be good for the facet generation as it reduces the size of the network at the same time as not a single connection is removed. The number of facets will decrease since the node removal is in fact a way of introducing more information to the process, namely that β(i) = 0 for all transshipment nodes. The effect of arc removal is more complicated. First, since also arc removal means introducing new information, the number of facets will decrease. That is good news. Second, the arc removal will decrease the number of elements in some F* and B*. That is bad news as it increases the complexity of the network, and hence the CPU time spent in the recursive part of the algorithm. Since few facets probably means few inequalities in terms of x to be added to the first-stage constraints, one therefore has to trade between the cost of obtaining the facets and the cost of having a large first-stage problem.

A more subtle point is that the arc removal should be applied only to those nodes that are in S* or D*, but which are not part in any collapse of directed circuits. Or to put it a different way, arc removal should be applied after the call to CollapseNodes, and only to nodes i ∈ S* or D* for which A(i) = {i}. If arc removal is applied before CollapseNodes, the node in question might be removed from a circuit, thereby introducing unnecessary complications into the network. Applying arc removal to a node where A(i) ≠ {i} is wrong, since it means using information that is not given.

The tests in this Section are performed on a SUN 3/50 with the same code as in Section 2. On these problems, the SUN 3/50 runs approximately 20% faster than the Prime 9950.
We have had some difficulties in generating difficult problems with NETGEN. It seems to have built into it some "nicety-properties." This is partly the reason why many of the examples in Section 2 have so few facets, in fact, many of the examples turned out to consist of directed circuits and trees attached to them. To illustrate the effect of arc removal we must come up with more difficult examples. In this section we have therefore done the following: First NETGEN was run in a normal fashion, then a process that removed every \( t \)-th arc, where \( t \) was uniformly distributed over \([0, 10]\), was employed. This was repeated until the network became disconnected. Due to this process, we have not presented the input data for NETGEN as we did in Section 2, since one would need the code that generated the realizations for \( t \) to replicate our results. However, let us point out that Example 5 below equals Example S in Section 2. Note that the examples below are in no way arbitrary, since we had to use some efforts to generate them. In a lot of cases we obtained examples that could not be used to illustrate the difficulties that may be encountered.

Below follows a table showing the effect of using arc removal after the call to CollapseNodes for those nodes in \( S^* \) and \( D^* \) that were not in any directed circuit. We have chosen to let \( S^* = \{1, \ldots, 25\} \) and \( D^* = \{n - 24, \ldots, n\} \). (NETGEN also defines its supply nodes to be the lowest numbered nodes and its demand nodes to be its highest numbered nodes.)

Table III should be read as follows. Example 1 has 250 nodes, 686 arcs, but after the collapse of directed circuits only 40 nodes are left. The two next columns show how we accumulate facets. After the call to TreeRemoval, 31 facets have been generated, and hence, 31 nodes have been removed. Then the facets corresponding to nodes in \( S \) and \( D \) are found, and we have a total of 35 facets. The total number of facets is 63. 0.2 CPU seconds were spent in the recursive part of the algorithm. Since, for complicated problems, the CPU time is determined by the time spent in the recursive part, we give only that number. The second line for Example 1 shows that the development is different for the case when \( S^* \) and \( D^* \) are used. There are fewer facets, and it took less time to find them. This, however, happens only for simple examples. For more complicated problems we still get a reduction in the number of facets, but it takes much more time to find them. Consider Example 5 (Example S in Section 2). The number of facets drops from 265 to 53, but the CPU increases from about 6 seconds to about 17 seconds.

The most extreme example is Example 3. Without the use of \( S^* \) and \( B^* \) we obtain 1072 facets in 6.4 CPU minutes. When using \( S^* \) and \( D^* \) the number of facets decreases to only 118, but it takes 9 CPU hours to find them. The question is, is it worthwhile to pay this extra cost in order to reduce the number of facets? Our answer is that it definitely might pay off. The reasons can be many. First, by having fewer facets, one will also expect to obtain fewer inequalities in terms of \( x \) (as explained at the end of Section 1) that need to be added to the first-stage constraints. This will pay off each time the master problem is solved. Second, the facets are independent of both \( b \) and \( d \). Hence, if the overall problem is solved again and again, but with changing \( b \) and \( d \), the facet generation needs to be performed only

<table>
<thead>
<tr>
<th>Ex.</th>
<th>No. of Nodes</th>
<th>No. of Arcs</th>
<th>Nodes after Collapse</th>
<th>Facets after TreeRemoval</th>
<th>Facets after ( S ) and ( D )</th>
<th>Total Number of Facets</th>
<th>CPU Recursion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>250</td>
<td>686</td>
<td>40</td>
<td>31</td>
<td>35</td>
<td>63</td>
<td>0.2 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>31</td>
<td>37</td>
<td>53</td>
<td>0.1 s</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>229</td>
<td>32</td>
<td>24</td>
<td>29</td>
<td>36</td>
<td>0.7 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>31</td>
<td>33</td>
<td>33</td>
<td>0 s</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>81</td>
<td>46</td>
<td>11</td>
<td>29</td>
<td>1072</td>
<td>6.4 m</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td>46</td>
<td>118</td>
<td>9 h</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>100</td>
<td>36</td>
<td>15</td>
<td>22</td>
<td>99</td>
<td>1.6 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>36</td>
<td>59</td>
<td>2.7 m</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>100</td>
<td>36</td>
<td>16</td>
<td>25</td>
<td>265</td>
<td>5.9 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>36</td>
<td>53</td>
<td>17.2 s</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>80</td>
<td>44</td>
<td>15</td>
<td>32</td>
<td>664</td>
<td>1.4 m</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td>45</td>
<td>176</td>
<td>46 m</td>
</tr>
</tbody>
</table>

*CPU times are in seconds (s), minutes (m) or hours (h) on a SUN 3/50. The results appear in pairs. The first row corresponds to the case where \( S^* \) and \( D^* \) is not used, the second to the case where they are used.
once. In such a case it might be optimal to pay a high cost once in order to avoid many small (or perhaps even large) costs later on. Third, one might be faced with a problem where information is made available at small time intervals. One therefore wishes to wait as long as possible to solve the stochastic optimization problem. By performing the preprocessing ahead of time, possibly at a high cost, the time needed to solve the problem will decrease, and hence one can wait longer before starting the main procedure.

Appendix A

The purpose of this appendix is to give a small example of the effect of using $F^*(i)$ and $B^*(i)$ instead of $F(i)$ and $B(i)$. Consider the example shown in Figure 3. For the six node network in the left hand side of Figure 3, the corresponding $F^*(i)$ and $B^*(i)$ are given in Table IV. The right part of Figure 3 shows how the network will be interpreted from the data in Table IV. For example, since $F^*(4) \setminus F^*(1) = \{4\}$, and $1 \in F^*(4)$ we know that all nodes that can be reached from node 4 can be reached via node 1. Hence, only one arc exits node 4. (The same can be seen from the fact that $B^*(1) \setminus B^*(4) = \{1\}$, and $4 \in B^*(1)$ so that all nodes that can reach node 1 can do it via node 4.) This way of interpreting a network is implicit in all the arguments in this paper.

Appendix B

The following is a description of input data to NETGEN (see Klingman et al.\cite{143}) as used in the numerical results in Section 3. For an explanation see the source text of NETGEN.

THE FIRST DATA CARD: a 0 in all examples.

CARD 1:
87654321 for A, D, G, S, T.
76543218 for B, E, H, M, N, O, P, Q, R, U.
65432187 for C, F, I, J, K, L.

CARD 2:
Total number of nodes: See Table I.
Number of arcs: See Table I.
Minimum cost for arcs: 1.
Maximum cost for arcs: 20.
Number of transshipment source nodes: A–C: 20, D–M: 15, N–O: 5, P–U: 0.
Number of transshipment sink nodes: A–C: 20, D–M: 15, N–O: 5, P–U: 0.
Percentage of skeleton arcs to be given the maximum cost: 100.
Percentage of arcs to be capacitated: 100.
Minimum upper bound: 1.
Maximum upper bound: 10.

Appendix C

The purpose of this appendix is to present a formal description of how we think a solution procedure, based on the L-shaped decomposition procedure, should be built up. For details the reader is referred to Van Slyke and Wets\cite{20} and Wets\cite{26}.

### TABLE IV

<table>
<thead>
<tr>
<th>Node</th>
<th>$F^*(i)$</th>
<th>$B^*(i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 4, 6}</td>
</tr>
<tr>
<td>2</td>
<td>{2, 3, 5}</td>
<td>{1, 2, 4, 6}</td>
</tr>
<tr>
<td>3</td>
<td>{3}</td>
<td>{1, 2, 3, 4, 6}</td>
</tr>
<tr>
<td>4</td>
<td>{1, 2, 3, 4, 5}</td>
<td>{4, 6}</td>
</tr>
<tr>
<td>5</td>
<td>{5}</td>
<td>{1, 2, 4, 5, 6}</td>
</tr>
<tr>
<td>6</td>
<td>{1, 2, 3, 4, 5, 6}</td>
<td>{6}</td>
</tr>
</tbody>
</table>

Figure 3. An example network (left) with its reduced form (right).
procedure L-shaped(\(\epsilon_1, \epsilon_2\) : scalars); begin 
\(\gamma := 0; \delta := 0;\) 
repeat 
repeat 
Master\((x, \theta, \gamma, \delta)\); 
FindLow\((x, L, \text{feasible})\); if not feasible then begin 
\(\delta := \delta + 1;\) 
FeasCut\((x)\); end else begin 
\(\gamma := \gamma + 1;\) 
OptCut\((x)\); end; 
until \(\theta \geq L - \epsilon_1;\) 
FindUp\((x, U)\); 
until \(U - L \leq \epsilon_2;\) end;

Let us now discuss the different parts of Procedure L-shaped.

### The Master Problem

In its simplest form, the procedure would be as follows.

```plaintext
procedure Master\((x : \text{vector}; \theta, \gamma, \delta : \text{scalars})\); begin 
if \(\gamma > 0\) then 
Solve min \(q_\nu x + \theta\), subject to 
\(Ax = a\) 
\(\theta \geq Z_i x + z_i\), \(i = 1, \ldots, \gamma\) 
\(0 \geq V_i x + u_i\), \(i = 1, \ldots, \delta\) 
\(x \geq 0\) 
else 
let \(\theta := -\infty\) and find a feasible \(x;\) end;
```

For a different approach to the master problem, see Ruszcynski.\(^{[19]}\)

### Feasibility Cuts

The purpose of a feasibility cut is to remove an \(x\) for which \(Q(x, b, d)\) is infeasible for some \(b\) from the set of feasible solutions to the master problem.

```plaintext
procedure FeasCut\((x : \text{vector})\); begin 
Create a cut of the type \(V x + v \leq 0;\) end;
```

For general discussions we refer to Wets.\(^{[26]}\) For details on the network problem we refer to Section 7.1 in Wallace.\(^{[21]}\)

### Optimality Cuts

The purpose of an optimality cut is to remove some \((\theta, x)\) which has proven not to be optimal from the set of feasible solutions to the master problem.

```plaintext
procedure OptCut\((x : \text{vector}; \theta : \text{scalar})\); begin 
Create a cut of the type \(Z x + z \leq \theta;\) end;
```

For general details see Wets.\(^{[26]}\) for details on the network case see Section 7.2 in Wallace.\(^{[21]}\)

### Lower Bounds and Upper Bounds

Unless the number of possible values of \((b, d)\) is very small, one cannot expect to be able to calculate \(\mathcal{E}(x)\). One then has to resort to approximations. Define the two functions \(\mathcal{L}(x)\) and \(\mathcal{U}(x)\) such that \(\mathcal{L}(x) \leq \mathcal{E}(x) \leq \mathcal{U}(x)\) where

\[
\mathcal{L}(x) = \int \int L(x, b, d) \ dP_L(b) \ dP_L(d)
\]

and

\[
\mathcal{U}(x) = \int \int U(x, b, d) \ dP_U(b) \ dP_U(d)
\]

The functions \(L(x, b, d)\) and \(U(x, b, d)\) are such that \(L(x, b, d) \leq Q(x, b, d) \leq U(x, b, d)\) for all relevant arguments. The distributions \(P_L(b)\), \(P_U(b)\), \(P_L(d)\) and \(P_U(d)\) are such that for all convex \(f\) and \(g\)

\[
\int f(b) \ dP_L(b) \leq \int f(b) \ dP_U(b) \leq \int f(b) \ dP_L(b)
\]

and

\[
\int g(d) \ dP_U(d) \leq \int g(d) \ dP_L(d) \leq \int g(d) \ dP_U(d)
\]

Most work on approximations has concentrated on finding approximate distributions, but some efforts have been made to approximate the function \(Q(x, b, d)\) itself. For an overview of approximation methods, see Birge and Wets.\(^{[5]}\) Later work includes that of Birge and Wets,\(^{[6]}\) Frauendorfer,\(^{[8]}\) Frauendorfer and Kall,\(^{[9]}\) Birge and Wallace\(^{[3,4]}\) and Wallace.\(^{[22]}\) Some of these methods approximate only \(Q(x, b, d)\), some only \(P(b)\) and \(P(d)\), whereas some do both. If all random variables are independent, the support is a high dimensional rectangle. The papers described above mostly present methods for finding \(L(x, b, d)\), \(U(x, b, d)\), \(P_L(b)\), \(P_U(b)\), \(P_L(d)\) and \(P_U(d)\) on a given such rectangle. If the result is not good enough, the
procedure Refine, used in the procedure L-shaped, partitions the rectangle into subrectangles, called cells. Some discussions about how to do that can be found in Birge and Wets,\textsuperscript{10} however, to a large extent this is an open question.

\begin{verbatim}
procedure FindLow(x: vector; L: scalar; feasible: boolean);
begin
  CheckFeas(x, feasible, FSET);
  if feasible then calculate $L = \mathcal{L}(x)$;
end;
\end{verbatim}

If it is computationally feasible, we let $\mathcal{L}(x) = \mathcal{E}(x)$. Otherwise one of the methods mentioned above will be needed.

\begin{verbatim}
procedure CheckFeas(x: vector; feasible: boolean; FSET: set of b realizations);
begin
  if $Q(x, b, d)$ is feasible for all $b \in FSET$ then
    feasible := true
  else
    feasible := false;
end;
\end{verbatim}

References for how to check feasibility for a given $b$ realization are the same as those for finding feasibility cuts. In addition, there is the question of how to determine the set $FSET$. This is a major question which has not been fully answered in the literature. We did address that question in Section 3 of this paper, for problems with (uncapacitated) network recourse.

\begin{verbatim}
procedure Refine;
begin
  refine the division of the support by splitting one or more cells.
end;

procedure FindUp(x: vector; U: scalar);
begin
  calculate $U = \mathcal{H}(x)$;
end;
\end{verbatim}

ACKNOWLEDGEMENTS

The authors wish to thank Bengt Aspvall for interesting and useful discussions on the use of transitive closures and strongly connected components in this paper.

Both authors are supported in part by NATO Collaborative Research Grant 0785/87. Wets is also supported in part by The Royal Norwegian Council for Scientific and Industrial Research.

REFERENCES


