Two-terminal Reliability Bounds based on Edge Packings by Cutsets

Louis D. Nel
School of Computer Science
Carleton University
Ottawa, Ontario K1S 5B6

Heidi J. Strayer
Department of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1

Abstract. A simple model of an unreliable network is a probabilistic graph in which each edge has an independent probability of being operational. The two-terminal reliability is the probability that specified source and target nodes are connected by a path of operating edges. Upper bounds on the two-terminal reliability can be obtained from an edge-packing of the graph by source-target cutsets. However, the particular cutsets chosen can greatly affect the bound. In this paper we examine three cutset selection strategies, one of which is based on a transshipment formulation of the k-cut problem. These cutset selection strategies allow heuristics for obtaining good upper bounds analogous to the pathset selection heuristics used for lower bounds. The computational results for some example graphs from the literature provide insight for obtaining good edge-packing bounds. In particular, the computational results indicate that, for the purposes of generating good reliability bounds, the effect of allowing crossing cuts cannot be ignored, and should be incorporated in a good edge-packing heuristic. This gives rise to the problem of finding a least cost cutset whose contraction in the graph reduces the source-target distance by exactly one.

Notation

$G = (V, E)$	Graph on node set V and edge set E.
$D = (V, E)$	Directed Graph on node set V and arc set E.
n, m	Number of nodes and edges of the graph respectively.
s, t	The source and target node of the graph respectively.
p_e	Operating probability of edge e.
$\text{cap}(e)$	Capacity of an edge e.
$S \subseteq E$	Network State: the subset of operating edges of the graph.
$r(s, t, S)$	binary connection function. 1 if s and t are connected by only edges in S, 0 otherwise.
$\text{Rel}(G, s, t)$	The two-terminal reliability of G with respect to nodes s and t.
$C_1, \ldots, C_k$	A set of k edge-disjoint cutsets of G.
$M_1, \ldots, M_k$	A set of k edge-disjoint pathsets of G.
$f(i, j)$	Amount of flow along an arc (i, j).
$\text{cost}(i, j)$	Cost per unit flow along an arc (i, j).

JCMCC 13 (1993), pp. 3-22

Introduction

A simple model of a communications network is a graph $G = (V, E)$ in which each edge $e \in E$ operates with an independent probability p_e. The two-terminal reliability $\mathrm{Rel}(G, s, t)$ of G with respect to a source node s and target node t is the probability that s and t are connected by a path of operating edges. A *state* of the network is a subset $S \subseteq E$ which represents precisely the operating edges. The probability $P(S)$ that the network is in state S is $\prod_{e \in S} p_e \prod_{e \in E-S}(1 - p_e)$. Define $r(s, t, S)$ to be 1 if s and t are connected in state S, and 0 otherwise. The two-terminal reliability $\mathrm{Rel}(G, s, t)$, can be expressed in terms of all possible network states as follows:

$$\mathrm{Rel}(G, s, t) = \sum_{S \subseteq E} P(S) r(s, t, S).$$

The number of possible network states is, in general, exponential in the size of the network, and computing the two-terminal reliability exactly known to be *NP-hard* [20], in other words, an amount of time which is exponential in the size of the network will be required, in general, to compute the reliability exactly. As a result, much research has been devoted to methods for estimating or approximating reliability. These methods either approximate the reliability by sampling or incomplete enumeration techniques, or by limiting the information extracted from the graph. Sampling or Monte Carlo techniques obtain an estimate of the reliability by sampling random network states. The accuracy of the estimate increases as more samples are taken. An excellent comparison of Monte Carlo-based approximations is described in [9]. Incomplete enumeration, such as the partial factoring described in [8], approximate the reliability by only partially completing an exact computation. One generally obtains upper and lower bounds from the partially completed exact computation. Both sampling and partial enumeration are restricted by the amount of time the problem solver is willing to spend on the problem. In contrast, methods such as cutset or pathset based methods, including those described in this paper, only use partial structure information. In this approach one approximates the problem by accounting for only some of the graph structure, and then solving the approximate problem *exactly*. These computations are not generally limited by the computation time, but rather the amount of information which is used to approximate the graph. In other words, the computations can usually be carried to completion in reasonable time, but accuracy of the estimates are limited by the amount of the actual graph structure which is accounted for. For example, in this paper the behavior of the network is approximated by accounting only for certain cutsets, as opposed to all possible failure modes.

A collection of subgraphs $G_1 = (V, E_1), G_2 = (V, E_2), \ldots, G_k = (V, E_k)$, in which $E_1, E_2, \ldots, E_k$ are all mutually disjoint subsets of E, is called an *edge-packing* of the graph $G = (V, E)$. In other words, it is a collection of edge-disjoint

subgraphs of G. An s,t-path or pathset is a minimal set of edges which connect s and t, an s,t-cutset or cutset is a minimal set of edges whose removal from the graph leave s and t disconnected. One obtains lower and upper bounds on the two-terminal reliability by edge-packing the graph with pathsets and cutsets, respectively. If $M_1, M_2, \ldots, M_k$ is a collection of k edge-disjoint s,t-paths, then

$$\mathrm{Rel}(G,s,t) \geq 1 - \prod_{i=1}^{k} \left(1 - \prod_{e \in M_i} p_e\right). \tag{1}$$

If $C_1, C_2, \ldots, C_k$ is a collection of k edge-disjoint s,t-cutsets, then

$$\mathrm{Rel}(G,s,t) \leq 1 - \prod_{i=1}^{k} \left(1 - \prod_{e \in C_i} (1 - p_e)\right). \tag{2}$$

Applying the above bounds to a particular edge-packing is straightforward, but these formulations say nothing about how the edge-packings (pathsets or cutsets) should be chosen in order to obtain the best bound. For example, one might select a maximum number of edge-disjoint paths, shortest paths, most reliable paths, or combinations of these. In general, the selection of paths or cutsets affects the bound. Nor is the strategy for selecting an optimal edge-packing straightforward. In particular, the problem of selecting a set of edge-disjoint s,t-paths which maximizes the lower bound is *NP-hard* [16].

Two-terminal lower bounds based on edge-packings by s,t-paths have been studied a great deal. This is well surveyed in two recent research monographs [5] and [19]. The most successful path selection strategy appears to be the *mincost* method of Brecht and Colbourn [3] [4]. It is a heuristic method which employs network flow techniques. If each edge $e \in E$ is assigned a capacity $\mathrm{cap}(e)$, a *flow assignment* assigns to each edge a flow $f(e)$ such that $0 \leq f(e) \leq \mathrm{cap}(e)$. Flows have direction; this is usually accommodated by transforming an undirected graph into a directed graph by replacing each edge with a pair of directed arcs. The flow $f(e)$ on arc e is flow in the direction of the arc. An s,t-flow is the net amount of flow from s to t. A *maximum* s,t-flow is an s,t-flow of maximum value. If each edge is also assigned a cost $\mathrm{cost}(e)$, the cost of a flow assignment is $\sum_{e \in E} f(e)\,\mathrm{cost}(e)$.

Network flows have been studied a great deal and some excellent presentations are given by Gondran and Minoux [12], Lawler [14], and Ford and Fulkerson [11]. A network in which each edge has a capacity of 1 and which has an s,t-flow with integer value f must have f edge-disjoint s,t-paths. The largest f can be is c, the size of a minimum cardinality s,t-cutset. Brecht and Colbourn assign to each edge a cost of $-\log(p_e)$ and compute a minimum cost flow assignment for each of the

integer flow values f between 1 and c. In each case this results in $k = f$ edge-disjoint s,t-paths $M_1 \ldots M_k$ which minimizes $\sum_{i=1}^{k} \sum_{e \in M_i} - \log(p_e)$ or equivalently maximizes $\sum_{i=1}^{k} \sum_{e \in M_i} -p_e$. The largest which results from among these edge-packings is what they call the *mincost* lower bound. The method is heuristic because the edge-packing $M_1, \ldots, M_k$ which maximizes $\sum_{i=1}^{k} \sum_{e \in M_i} -p_e$ does not in general maximize (1); but it appears to be an effective strategy. Brecht [3] has reported on the relative performance of different two-terminal lower bounds (including the Kruskal-Katona bounds) and observes that the *mincost* lower bound is the best in the majority of his test cases.

The Kruskal-Katona bound is the other main combinatorial method for computing reliability bounds. It is based on a reliability polynomial in which coefficients of the polynomial count the states of a particular size in which s and t are connected. The Kruskal-Katona bounds, however, apply only to graphs in which each edge operates with the same probability. Computational results on example graphs in [3] [5] suggest that, even under this assumption, the edge-packing lower bounds are better than the Kruskal-Katona bounds in the two-terminal problem.

Edge-packing upper bounds have not been studied to the same extent. In this paper we describe three strategies for selecting s,t-cutsets and compare the two-terminal upper bounds that result.The first strategy always selects a maximum number of edge-disjoint cutsets. The second selects minimum capacity cutsets in a greedy manner using network flows. Finally, Wagner [21] has recently shown that the k-cut problem (defined in the next section) can be formulated as a transshipment problem, and hence can be solved in polynomial time. The solution to the k-cut problem provides a bounding strategy for upper bounds which is analogous to the mincost strategy for lower bounds. A brief synopsis of the paper follows.

In Section 1 we present the first two selection algorithms, and introduce the k-cut problem. In Section 2 we review the steps of the network simplex method. In Section 3 Wagner's k-dicut transshipment formulation is described along with a graph theoretic implementation for finding k edge-disjoint s,t-cutsets of minimum cost. Finally, in Section 4 we apply these bounding strategies to some example graphs from the literature. These results provide insight into the requirements of a good cutset selection heuristic, and gives rise to a new minimum cost cutset selection problem.

1.0 Selecting Edge-Disjoint Cutsets

The maximum number of edge-disjoint s,t-cutsets a graph can have is equal to the length of a shortest s,t-path [17]. A maximum number of edge-disjoint s,t-cutsets can be found easily by performing a breadth-first search starting from the source node s. Let l be the length of a shortest s,t-path in G. Let $V_i, i = 1, \ldots, l$ be the set of nodes reachable from s along a path of length i. Let E_i be the set of all edges

between a node in V_i and V_{i-1}. Each E_i contains a unique s,t-cutset which in edge-disjoint from that of $E_j, j \neq i$. This appears to be the first selection strategy used for computing two-terminal upper bounds based on edge-packings by cutsets [6] [10]. The resulting bounds reported in [6] for the graph in Figure 4.1 are competitive with the subgraph counting Kruskal-Katona bounds, notwithstanding the fact that the Kruskal-Katona bounds apply only when the edges probabilities are the same.

Recall that for an edge-packing $C_1, \ldots, C_k$ by s,t-cutsets, the two-terminal upper bound is given by equation (2). It is evident from this formulation that a large number of small cutsets would tend to keep the upper bound low. Nevertheless, selecting a maximum number of cutsets often results in relatively large cutsets (in part as a result of the fanout effect of a breadth-first search). Large cutsets work against keeping the upper bound low. This suggests using an edge-packing with small, failure prone s,t-cutsets, and not necessarily a maximum number of s,t-cutsets. We again employ the theory of network flows.

Instead of selecting an edge-packing with a maximum number of cutsets, the idea here is to select a cutset C for which the factor $1-\sum_{e\in C}(1-p_e)$ is minimum. This is equivalent to selecting a cutset C for which $\sum_{e\in C}(1-p_e)$ is a maximum. If all edges operate with the same probability p, this is equivalent to selecting a minimum cardinality s,t-cutset; it need not be a minimum cardinality cut if the edge probabilities differ.

Maximizing $\prod_{e\in C} 1-p_e$ is equivalent to minimizing $-\log\left(\sum_{e\in C}(1-p_e)\right)$. If we assign to each edge e a capacity of $-\log(1-p_e)$ the problem becomes a linear minimization problem, namely, finding an s,t-cutset C which minimizes $\sum_{e\in C} -\log(1-p_e)$; in other words, finding a minimum capacity cutset. Again, the maximum network flow problem can be solved efficiently, and as a side-effect produce a minimum capacity cutset. See for example Hu [13]. The complete edge-packing is obtained by selecting an s,t-cutset C which minimizes $1-\sum_{e\in C}(1-p_e)$, contracting the edges of this cutset, and repeating this process until no further edge-disjoint cuts exist. Contracting an edge involves deleting the edge and collapsing its two end nodes into a single node.

Nel and Colbourn [15] used this greedy selection of minimum capacity cutsets to obtain upper bounds for the graph in Figure 4.1 and observed that the resulting bounds are better than those obtained using the breadth-first search for more than 99% of all the possible s,t pairs.

The last selection strategy is based on a transshipment formulation of the k-cut problem. Given a graph $G=(V,E)$ with distinguished nodes s, and t, and a real valued cost cost(e) associated with each edge $e \in E$, the *k-cut* problem is to find k edge-disjoint s,t-cutsets $C_1, \ldots, C_k$ which minimizes $\sum_{i=1}^{k}\left(\sum_{e\in C_i}\text{cost}(e)\right)$. If each edge is assigned a cost of $-\log(1-p_e)$, an edge-packing which solves the k-cut problem minimizes $\sum_{i=1}^{k}\sum_{e\in C_i} -\log(1-p_e)$ or equivalently, maximizes

$\sum_{i=1}^{k} \sum_{e \in C_i} (1 - p_e)$. Again maximizing $\sum_{i=1}^{k} \sum_{e \in C_i} (1 - p_e)$ may not minimize $\sum_{i=1}^{k} \left(1 - \sum_{e \in C_i} (1 - p_e)\right)$, but is a heuristic analogous to the mincost pathset selection for the lower bounds. For $k = 1$ the problem reduces to the minimum capacity cutset problem described above. When $k = c$ a maximum number of edge-disjoint cutsets is selected, but the bounds may differ from those obtained using a maximum number of cutsets selected by breadth-first search.

Wagner [21] has recently shown that the k-cut problem can be formulated as a transshipment problem, and hence can be solved in polynomial time. Transshipment problems are special cases of linear programming problems and can be solved very elegandy with the network simplex method. In the next section we briefly review the steps in the network simplex method [7]. In Section 3 we illustrate the solution of the k-cut problem using this simplex method. The transformations and algorithms are illustrated in a graph-theoretic context (as opposed to a linear programming context and lead to easy computer implementation. For a linear programming formulation, as well as correctness proofs for the transformations see [21].

2.0 The Network Simplex Method

The network simplex method is a well known method for solving transshipment problems; for an excellent general description see Chvátal [7]. We review the basic steps of the network simplex method to clarify the transshipment formulation of the k-cut problem in the next section. Readers familiar with transshipment problems and the network simplex method can skip to the next section without loss of continuity.

An instance of a transshipment problem is a directed graph $D = (V, E)$ in which each node has an associated demand (negative demands indicated sources, and positive demands indicate sinks) and each arc $(i, j) \in E$ has an associated cost $\text{cost}(i, j)$. The objective is to assign a non-negative amount of flow $f(i, j)$ to each arc $(i, j) \in E$ such that the net flow to each node equals its demand, and the cost of the flow assignment $\sum_{(i,j) \in E} f(i, j)\text{cost}(i, j)$ is minimized. In addition, it is usually assumed that the total supply equals total demand (i.e. the sum of the node demands is zero). Transshipment problems can be solved very elegantly with the network simplex algorithm. We illustrate the steps involved using the following example from [7].

In figure 2.1 the arc labels are the arc costs and the node demands for nodes $v_1, \ldots, v_7$ are 0, 0, 6,10, 8, -9, and -15, respectively. A feasible tree solution for a transshipment problem is a flow assignment which meets the flow constraints, and in addition, the subgraph consisting of all arcs with non-zero flow is a forest. Arcs with 0 flow are usually added to complete a spanning tree. A feasible tree solves the transshipment problem if it also minimizes the objective function. If a transshipment problem has a feasible solution, it has a feasible tree solution and, moreover, has an optimal feasible tree solution [7]. The network simplex

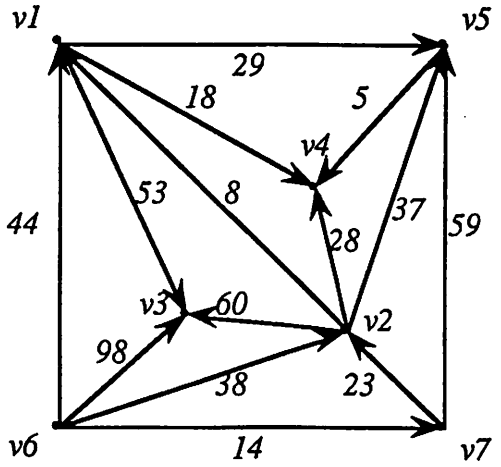

Figure 2.1

algorithm operates exclusively on feasible tree solutions and iteratively refines them until an optimal tree results.We illustrate this starting with the following feasible tree solution; general methods for finding initial solutions are discussed in [7].

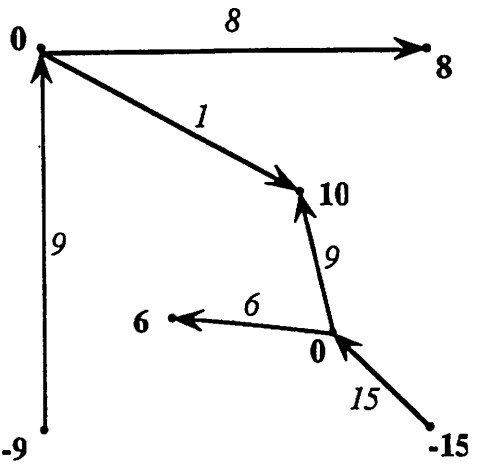

Figure 2.2

In figure 2.2 the edge labels are the assigned flows; the node labels are the demands. It is easy to verify that this solution meets the flow constraints. The

network simplex method iterates with the following steps.

Step 1: Starting with a reference node which is assigned a price of 0, determine "fair prices" y_j for each remaining node j as follows. $y_j = y_i + \text{cost}(i,j)$ for (i,j) an arc in the current feasible tree. For example, selecting node 7 as the reference node and setting $y_7 = 0$ we obtain the following node prices (figure 2.3).

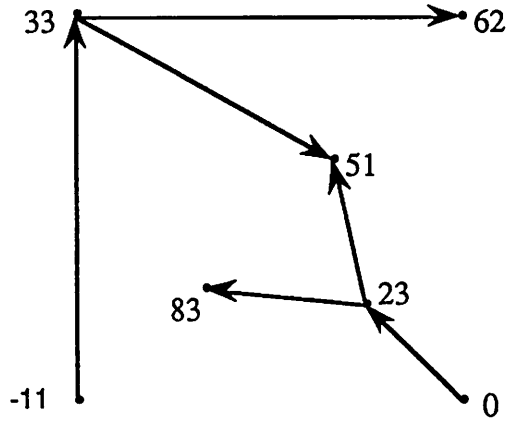

Figure 2.3

Step 2: Select an entering arc. Choose a non-tree arc (i,j) such that $y_i + \text{cost}(i,j) < y_j$. If no such arc exits the current tree is optimal and we can stop. For example, choosing arc $(7,5)$ as the entering arc, in which case $y_7 + \text{cost}(7,5) = 59 < y_5$, we obtain figure 2.4

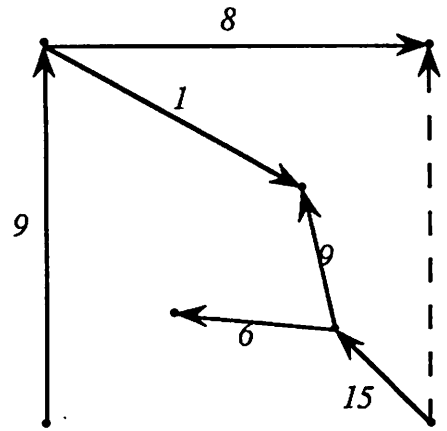

Figure 2.4

Step 3: . Select a leaving arc (pivot step). Examine the unique cycle formed by the entering arc and the current feasible tree. Define a *forward* traversal of this cycle to be in the direction of the entering arc, and *reverse* otherwise. Identify the smallest flow assignment f from among the *reverse* arcs in the cycle, in this case $f = 8$ on arc $(1,5)$. Subtract this amount of flow f from all reverse arcs in the cycle and add this amountf to all the forward arcs. This defines a new feasible tree solution, as shown in figure 2.5.

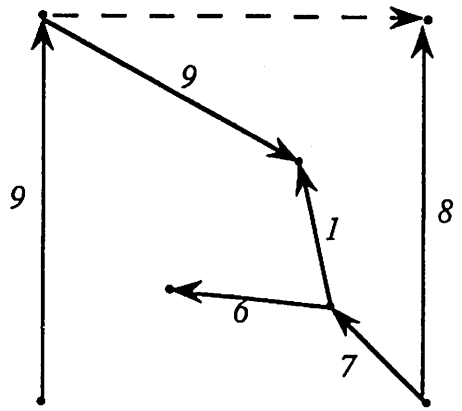

Figure 2.5

These three steps are iterated until an optimal soludon results.

3.0 The Dicut Transshipment Formulation

Recall that the k-cut problem is to find k edge-disjoint s,t-cutsets $C_1,\ldots,C_k$ which minimizes $\sum_{i=1}^{k}\left(\sum_{e\in C_i}\text{cost}(e)\right)$. In the case of a directed graph $D = (V,E)$ call an arc $(i,j) \in C$ of an s,t-cutset C a *forward* arc if its tail is in the component of the subgraph $D\backslash C$ which contains the source node s, and call it a *reverse* arc otherwise. Define $F(C)$ to be the set of forward arcs of C. The directed k-cut problem is that of finding k edge-disjoint s,t-cutsets $C_1,\ldots,C_k$ which minimizes $\sum_{i=1}^{k}\left(\sum_{e\in F(C_i)}\text{cost}(e)\right)$. The directed k-cut problem includes the undirected k-cut problem under the following transformation. Replace each edge $e = (i,j)$ in the undirected graph with a pair of arcs (i,j) and (j,i) and assign to each arc the same cost $\text{cost}(e)$.

An s,t-cutset is an s,t-dicut if all its arcs are forward arcs. The k-dicut problem is that of finding k edge-disjoint s,t-dicuts $C_1,\ldots,C_k$ which minimizes $\sum_{i=1}^{k}$ $\left(\sum_{e\in C_i}\text{cost}(e)\right)$. The k-dicut problem includes the directed k-cut problem (and hence the undirected problem) under the following transformation. Replace each

arc (i,j) by two new arcs (i,w) and (j,w), where w is a new node incident only with these two arcs. Assign cost(i,w) = cost(i,j) and cost$(j,w) = 0$.

The main result of Wagner [21] is that the k-dicut problem (and hence the k-cut problem) can be formulated as a specially structured transshipment problem, and that the solution to the transshipment problem can be transformed into the required k-cut edge-packing. In what follows we illustrate this transformation from an undirected k-cut problem to an instance of the transshipment problem, and its solution using the network simplex method. Again, the description is graph-theoretic and is easily implemented on a computer. We illustrate the steps involved using the very simple instance of an undirected 1-cut problem ($k = 1$) in figure 3.1. This instance consists of a graph with two edges having costs of 10 and 1.

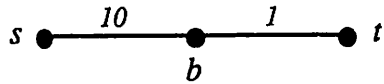

Figure 3.1

Step 1: Transform the undirected problem into an instance of a directed problem by replacing each edge with a pair of oppositely directed arcs (figure 3.2).

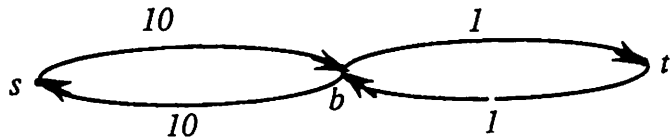

Figure 3.2

Step 2: Transform the directed instance into an instance of the k-dicut problem by replacing each arc (i,j) by a pair of arcs (i,w) and (j,w), introducing a new node w incident only with these two arcs. In addition, add a directed path P of length k from s to t but otherwise disjoint from the graph, and assign each edge of this path a cost of 0 (figure 3.3).

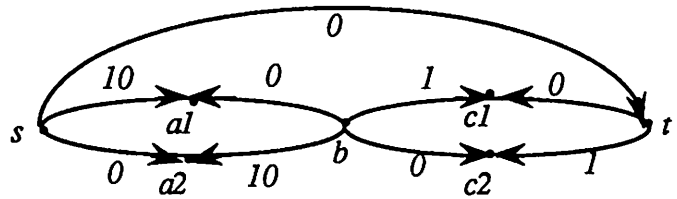

Figure 3.3

Step 3: Construct the transshipment graph from the previous graph as follows. Each node is assigned a demand equal to the sum its incoming arc costs minus its outgoing arc costs. The arc costs are all replaced by 0, except the arcs of the

added s,t-path which are each assigned a cost of -1. Finally, for every arc (i,j), add a new arc (j,i) with a cost of 1. The resulting graph, shown below, forms the instance of the transshipment problem. The transhipment formulation transforms a graph $G = (V,E)$ into a new graph $G' = (V',E')$ with $|V'| = |V| + 2|E|$ and $|E| = 2(4|E| + k)$. Thus while the size of the graph is increased it is clearly a polynomial transformation. The resulting graph appears in figure 3.4.

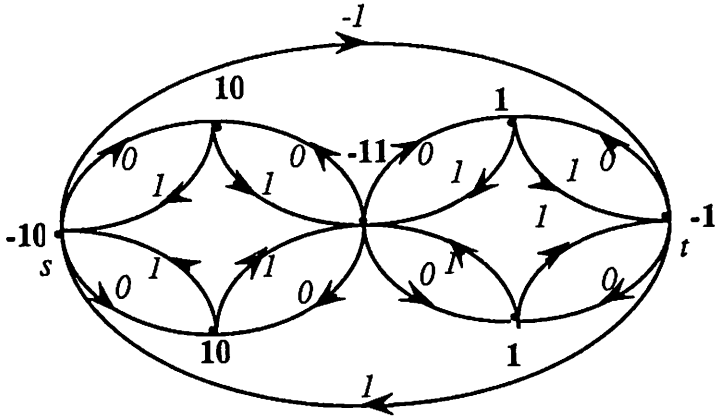

Figure 3.4

The next few figures illustrate the iterations of the network simplex method for this instance of the transshipment problem. Because of the special structure of the dicut transshipment problem an initial feasible tree solution is easily constructed from the intermediate graph obtained in Step 2 above.

i) Arcs with non-zero costs are included and assigned their cost.
ii) The added s,t-path is included (this cannot form a cycle).
iii) Other zero-cost arcs are added to complete a spanning tree of the graph (this ensures that an entering arc does, in fact, create a cycle).

(For a more general discussion on forming initial tree solutions see [7].) This results in the initial feasible tree solution in figure 3.5; nodes are labeled with their demands, and arcs are labeled with their assigned flow.

The first iteration of the network simplex method goes as follows. i) starting with the source node s, nodes are priced using the edge costs of the original transshipment graph. The resulting prices are shown on the node labels in figure 3.6. ii) An entering arc is chosen; shown by the dashed arc. iii) The arc with the smallest reverse flow is chosen as the leaving arc; shown by the double arrowhead.

Once the leaving arc is chosen the flow assignments are adjusted. The second iteration results in the following feasible tree (figure 3.7).

No further iterations are possible and so the feasible tree solution in figure 3.8 is optimal.

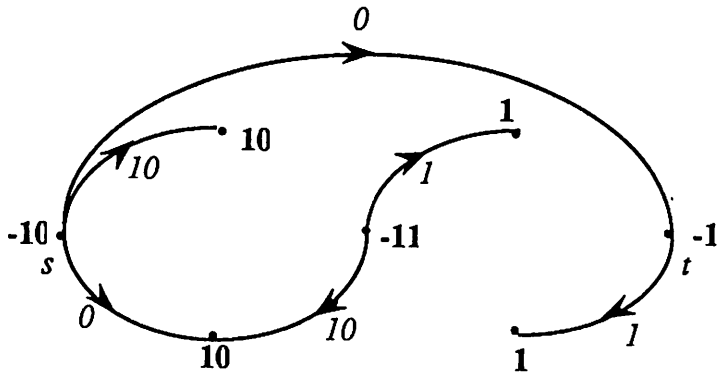

Figure 3.5

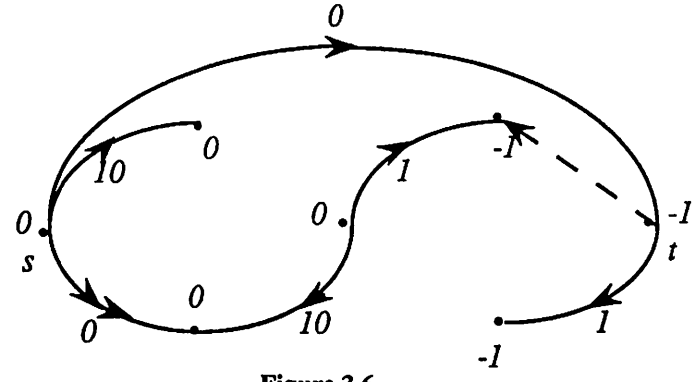

Figure 3.6

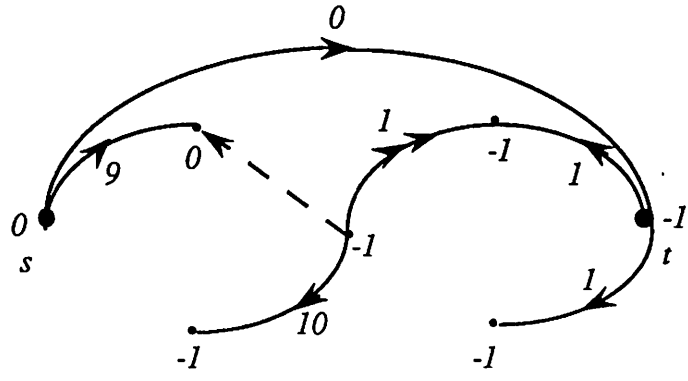

Figure 3.7

The final requirement is to transform the optimal tree solution of the transshipment problem into the desired edge-packing for the original k-cut problem. Let T be the resulting optimal tree. Define the *weight* of the unique i, j-path in T to

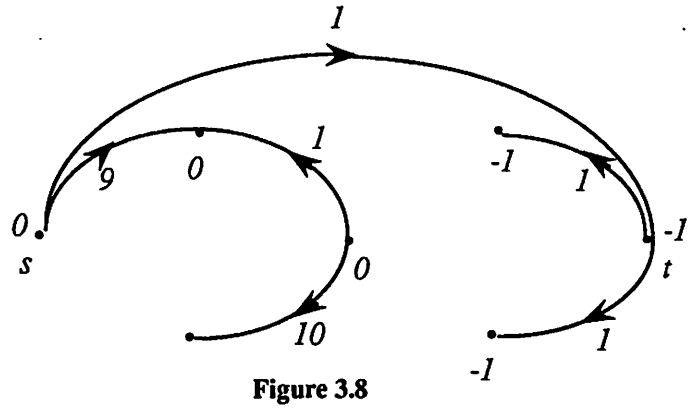

Figure 3.8

be the sum of the forward arc costs minus the sum of the reverse arc costs. Define $\Pi_s = 0$ and Π_i to be the weight to the unique s,i-path in T. Next, partition the node set V into r classes $V_1,\ldots,V_r$ such that V_i contains all nodes with the same weight Π, and V_1 contains the nodes with the largest n values, V_2 the second largest, and so on. Define K_i to be the set of all arcs leaving V_i. The sets $K_1,\ldots,K_r$ are disjoint dicuts with the property that exactly k of the dicuts are s,t-dicuts [21]. Moreover, each of these k dicuts contains exactly one of the k arcs from the s,t-dipath P added to the graph. The arcs of these s,t-dicuts are easily associated with edges of the original graph by reversing the transformation process used to create the transshipment graph from the original graph. In the optimal tree solution above each node is labeled with its Π value. The resulting partition is $V_1 = \{s, a1, a2, b\}$, and $V_2 = \{t, c1, c2\}$, which corresponds to cutset $\{(s,t), (b,c1), (b,c2)\}$ in the dicut graph (step 2) or cutset $\{(b,c)\}$ in the original undirected graph.

4.0 Computational Experience

In this section we apply the three cutset selection strategies to some example graphs from the literature. The first example, shown in Figure 4.1, is a skeleton of the American ARPA computer network which was used in the original applications of the breadth-first search method in [6], and again in [15]. The second example, shown in Figure 4.2, has been used by Fishman [9] for comparing Monte Carlo methods for estimating reliability. The accompanying tables, Table 1 and Table 2, show the upper bounds on the two-terminal reliability obtained using the three edgepackings for a variety of source-target pairs. In each case the bounds are shown for a range of edge operation probabilities. In these experiments all the edges in the graph were assigned the same operation probability, but this is not a restriction of the method itself.

Two observations are apparent from Tables 1 and 2. First, the breadth-first search method gives distinctly inferior bounds. Second, the minimum capacity

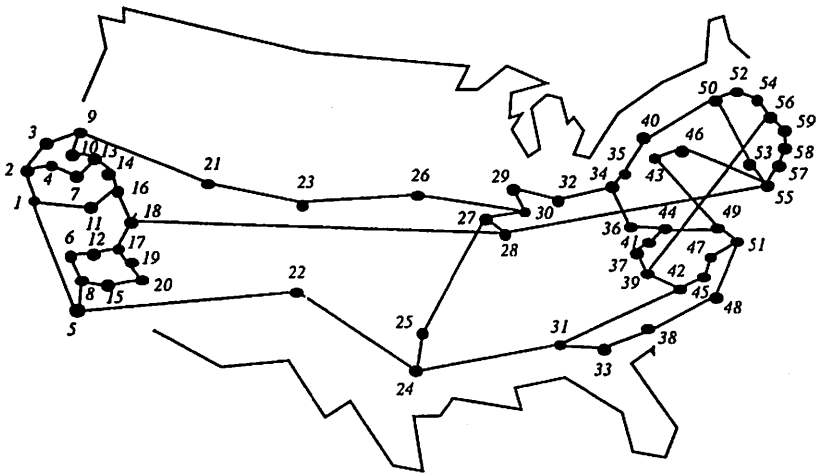

Figure 4.1

and k-cut methods give the same results in most cases, but where they do differ in the first example, the minimum capacity bounds are often much better. (The places in Table 1 where the minimum capacity and the k-cut results differ are marked with an *.) However, as the following small examples show, neither method is uniformly better than the other. In the figure 4.3.

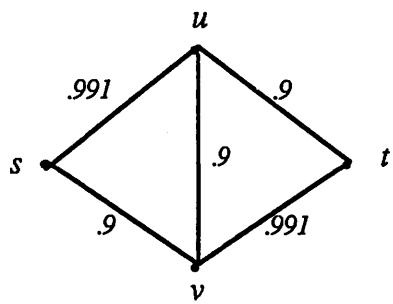

Figure 4.3

the minimum capacity method would select the cuts $\{(s,v),(u,v),(u,t)\}$, and then no further cuts are possible. The resulting two-terminal upper bound for this

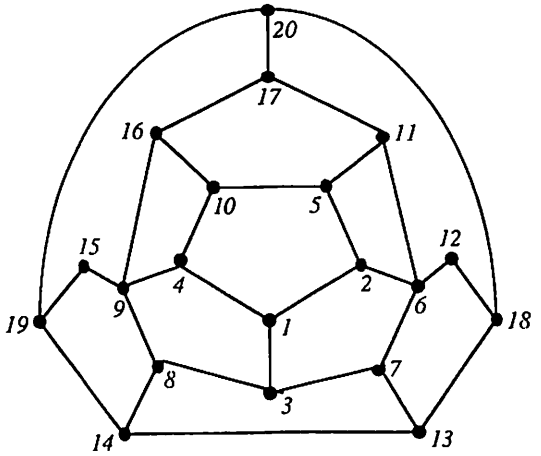

Figure 4.2

cutset is $1-(.1)(.1)(.1) = .999$.

On the other hand, for $k = 1$, the k-cut method would also select $\{(s,v),(u,v),(u,t)\}$, producing the same bound, but for $k = 2$ would select $\{(s,u),(s,v)\}$ and $\{(u,t),(v,t)\}$ which gives a bound of $(1-(.009)(.1))(1-(.009)(.1)) = .998201$. In this example the k-cut selection strategy gives the better bound.

In the following example (figure 4.4)

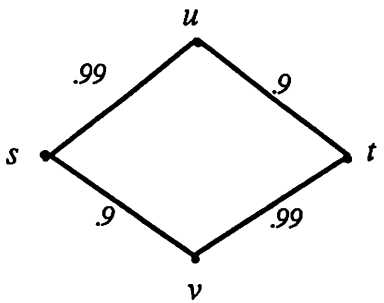

Figure 4.4

the minimum capacity method would select the cut $\{(s,v),(u,t)\}$ and $\{(s,u),(v,t)\}$, which gives the bound $(1-(.1)(.1))(1-(.01)(.01)) = .989901$. On the

Table 1: Upper bounds for ARPANET (Figure 4.1)				
s,t pairs	edge prob.	breadth-first search	minimum capacity	k-cut
1,57	.1	.005313	.000678*	.000858
	.3	.316912	.100855*	.109906
	.5	.773118	.467359*	.471039
	.7	.961798	.831290*	.831472
	.9	.998889	.986934	.986934
	.95	.999868	.997120	.997120
	.99	.999999	.999897	.999897
20,41	.1	.000010	.000002*	.000003
	.3	.025465	.012426*	.013551
	.5	.263124	.198977*	.209805
	.7	.672467	.629237*	.631690
	.9	.960393	.957706*	.957717
	.95	.990025	.989666	.989666
	.99	.999600	.999597	.999597
22,23	.1	.000646	.000332	.000332
	.3	.095321	.063725	.063725
	.5	.440022	.359201	.359201
	.7	.796691	.739611	.739611
	.9	.979011	.970095	.970095
	.95	.994875	.992506	.992506
	.99	.999799	.999700	.999700
22,34	.1	.002543	.000260*	.000277
	.3	.246897	.083277*	.087981
	.5	.670065	.456794*	.473635
	.7	.900914	.832166*	.838262
	.9	.989899	.987003*	.987033
	.95	.997494	.997125*	.997126
	.99	.999900	.999897	.999897
29,31	.1	.005154	.001965	.001965
	.3	.225704	.139170	.139170
	.5	.620920	.521507	.521507
	.7	.882441	.852469	.852469
	.9	.988999	.987912	.987912
	.95	.997375	.997244	.997244
	.99	.999899	.999898	.999898
42,53	.1	.005313	.001592	.001592
	.3	.316912	.164916	.164916
	.5	.773118	.577529	.577529
	.7	.961798	.874847	.874847
	.9	.998990	.996006	.996006
	.95	.999868	.997369	.997369
	.99	.999999	.999899	.999899

other hand, for $k = 1$ the k-cut method would select $\{(s,v),(u,t)\}$, which gives a bound of .990000, and for $k = 2$ the cuts would be $\{(s,u),(s,v)\}$ and $\{(u,t),(v,t)\}$ which gives a bound of $(1-(.01)(.1))(1-(.01)(.1)) = .998001$. In this example the minimum capacity method produces the better bound.

This second example is instructive because it accounts for the discrepancy between the minimum capacity results and the k-cut results in Table 1. In particular, the same total collection of edges are selected by each method to obtain two

Table 2: Upper bounds for Figure 4.2				
s,t pairs	edge prob.	breadth-first search	minimum capacity	k-cut
11,19	.1	.059497	.034411	.034411
	.3	.511503	.380866	.380866
	.5	.847870	.753662	.753662
	.7	.971582	.946039	.946039
	.9	.998998	.998000	.998000
	.95	.999875	.999750	.999750
	.99	.999999	.999998	.999998
14,20	.1	.126979	.073441	.073441
	.3	.579705	.431649	.431649
	.5	.861328	.765625	.765625
	.7	.972291	.946729	.946729
	.9	.998999	.998001	.998001
	.95	.999875	.999750	.999750
	.99	.999999	.999998	.999998
12,15	.1	.027878	.016124	.016124
	.3	.451325	.336057	.336057
	.5	.834622	.741886	.741886
	.7	.970874	.945349	.945349
	.9	.998997	.997999	.997999
	.95	.999875	.999750	.999750
	.99	.999999	.999998	.999998
1,20	.1	.007555	.007555	.007555
	.3	.296521	.296521	.296521
	.5	.730294	.730294	.730294
	.7	.944660	.944660	.944660
	.9	.997998	.997998	.997998
	.95	.999750	.999750	.999750
	.99	.999998	.999998	.999998
2,9	.1	.059497	.034411	.034411
	.3	.511503	.380866	.380866
	.5	.847870	.753662	.753662
	.7	.971582	.946039	.946039
	.9	.998998	.998000	.998000
	.95	.999875	.999750	.999750
	.99	.999999	.999998	.999998
6,9	.1	.027878	.016124	.016124
	.3	.451325	.336057	.336057
	.5	.834622	.741886	.741886
	.7	.970874	.945349	.945349
	.9	.998997	.997999	.998000
	.95	.999875	.999750	.999750
	.99	.999999	.999998	.999998

edge-disjoint cutsets. However, it is the way the edges are grouped into individual cutsets which affects the value of the resulting bound. Let us elaborate. For a particular value of k, the k-cut method selects a collection K of edges from the original graph with the property that these edges can be partitioned into k edge-disjoint s,t-cutsets. It is the cost of the entire collection of edges which is minimized, and not the cost of individual cutsets. Consider the graph G_1 obtained from the original graph G by contracting all of the edges in $E - K$. The length

of a shortest s,t-path in G_1 is precisely k and, moreover, every edge in G_1 is on a shortest s,t-path. In other words, if the edges in G_1 are partitioned into k edge-disjoint s,t-cutsets, every edge participates in exactly one cut. One obtains Wagner's partition of K into cutsets by applying the breadth-first search method to the resulting graph G_1. However, the partition of K into k edge-disjoint cutsets is not, in general, unique. In the last example the graphs G and G_1 are, in fact, the same and for $k = 2$ the minimum capacity and k-cut methods partition the same collection of edges into different cuts. The distinction is that the cutsets obtained from the k-cut method are *non-crossing*, whereas the cutsets obtained using the minimum capacity method can *cross*. Deleting a minimal s,t-cutset partitions a graph into two components, one containing s and the other containing t. Two s,t-cutsets are non-crossing if all the edges of one is contained within one of the components obtained by deleting the other; otherwise, the cutsets are crossing. The difference in the minimum capacity and k-cut bounds in Table 1 is accounted for by using the minimum capacity method to repartition the complete set of edges selected by the k-cut method.

The reliability bound given by equation (2) does not require that the cutsets be non-crossing, only that they be edge-disjoint. The restuls in Table 1 show that the effects of allowing crowssing cuts are significant. While the differences for the example in Table 1 are accounted for by repartitioning the k-cut edges using the minimum capacity method, there is no guarantee that the minimum capacity method would find a maximum number of cutsets. This is demonstrated in the first of the previous two examples, where greedy selection of the cutset makes further selection of cutsets impossible.

The previous discussion begs the question of whether or not the edge-set obtained from the k-cut method can be partitioned into k edge-disjoint s,t-cutsets by a method other than breath-first search. In particular, it gives rise to the following cutset selection problem.

Given a graph $G = (V, E)$ with distinguished nodes s and t, and real costs cost(e) assigned to every edge e, find a least cost s,t-cutset C such that if all the edges of C are contracted, the length of a shortest s,t-path decreases by exactly one. Again, the cost of the cutset is $\sum_{e \in C} \text{cost}(e)$. An efficient algorithm for this problem would provide an alternative partitioning heuristic and would ensure a maximum number of cutsets. Such an algorithm would have immediate application in computing reliability bounds.

Finally, there are situations in which non-crossing cutsets may be desirable. AboElFotoh [1] has recently developed a heuristic method for computing two-terminal reliability bounds which is based on forming series-parallel approximations of the original graph. His method makes an improvement over the edge-packing bounds in as much as it starts with, and preserves, an edge-packing by s,t-cutsets. However, a restriction of his approach is that the cutsets must be non-crossing. Thus the k-cut selection strategy, which ensures non-crossing cuts, has

immediate application to his approach.

One way to improve the reliability bound is to use more cutsets by allowing cutsets to overlap. That is, removing the restriction that the cutsets be edge-disjoint. Shanthikumar [18] has used this approach to obtain bounds based on *consecutive* minimal s,t-cutsets. Consider an ordered collection of edge cutsets $C_1 \leq C_2 \leq, \ldots, C_n$. The ordered collection of cutsets is consecutive if an edge belonging to cutsets C_1 and $C_2, C_1 \leq C_2$, also belongs to any cutset C, $C_1 \leq C \leq C_2$. Shanthikumar obtains a collection of consecutive s,t-cutsets as follows. Nodes are labeled $1, \ldots, n$, with $s = 1$ and $t = n$. The cutsets are C_i, $i = 1 \ldots n-1$, where C_i contains the unique s,t-cutset contained in the edges between the nodes $\{1, \ldots, i\}$ and $\{i+1, \ldots, n\}$. This collection of cutsets will, in general, be greater than the maximum number of edge-disjoint cutsets, and the probability that at least one cutset fails can be computed easily if the cutsets are consecutive [18]. One minus the probability that a least one cutset fails is precisely the upper reliability bound.

The disadvantage of using these consecutive cutsets is that the particular cutsets selected depends on the ordering imposed on the nodes. In particular a cutset whose contribution to the bounds is greatest may be missed and, in general, no crossing cuts are allowed. This suggests using the edge-packing methods as a way of assigning the node ordering in Shanthikumar's approach. For example, if the nodes are labelled according to the k-cut partitions, the consecutive set of cuts will include all the cutsets selected by the k-cut method.

5.0 Conclusions

We have examined three strategies for edge-packing a graph with s,t-cutsets, along with the reliability bounds which result. These selection strategies make possible two-terminal upper bounds which are analogous to the two-terminal lower bounds obtained from edge-packings by s,t-paths. Wagner's transshipment formulation of the k-cut problem makes possible a powerful heuristic for constructing an edge-packing by non-crossing s,t-cutsets. However, the computational results indicate that, for the purposes of generating good reliability bounds, the effect of allowing crossing cuts cannot be ignored, and should be incorporated in a good edge-packing heuristic. This gives rise to the problem of finding a least cost cutset whose contraction in the graph reduces the source-target distance by exactly one.

Acknowledgment

We would like to thank Charlie Colbourn for his valuable contribuations and assistance in writing this paper.

References

1. H.M.F AboElFotoh, *Reliability of Radio Broadcast Networks: A Graph The-*

oretic Approach, Ph.D. Dissertation, University of Waterloo, Dept. of Computer Science (1988).
2. F. Beichelt and L Spross, *Bounds on the Reliability of Binary Coherent Systems*, IEEE Trans. Rel. 38, No. 4 (1989), 425–427.
3. T.B. Brecht, *Lower Bounds for Two-Terminal Network Reliability*, M. Math thesis, Dept. of Computer Science, University of Waterloo (1985).
4. T.B. Brecht and C.J. Colbourn, *Lower Bounds for Two-Terminal Reliability*, Discrete Applied Math. 21(3) (1988), 185–198.
5. C.J. Colbourn, "The combinatorics of Network Reliability", Oxford University Press, 1987.
6. C.J. Colbourn, *Edge-Packing of Graphs and Network Reliability*, Discrete Math. 72 (1988), 49–61.
7. V. Chvátal, "Linear Programming", W.H. Freeman and Company, 1983.
8. T.A. Feo and R. Johnson, *Partial Factoring: An Efficient Algorithm for Approximating 2-Terminal Reliability on Complete Graphs*, IEEE Trans. Rel. 39, No. 3 (1990), 290–295.
9. G.S. Fishman, *A Comparison of Four Monte Carlo Methods for Estimating the Probability of s,t-Connectedness*, IEEE Transactions on Reliability R-35 2 (1986), 145–155.
10. V.A. Kaustov, YE.I. Litvak, and I.A. Ushakov, *The Computational Effectiveness of Reliability Estimates by Method of Nonedge-Intersecting Chains and Cuts*, Soviet Journal of Computer and System Sciences 24 (1986), 59–62.
11. L.R. Ford and D.R. Fulkerson, "Flows and Networks", Princeton University Press, 1962.
12. M. Gondran and M. Minoux, "Graphs and Algorithms", Wiley-Interscience, 1984.
13. T.C. Hu, "Combinatorial Algorithms", Addison-Wesley, 1982.
14. E.L. Lawler, "Combinatorial Optimization: Networks and Matroids", Holt Rinehart and Winston, 1976.
15. L.D. Nel and C.J. Colbourn, *Locating a Broadcast Facility in an Unreliable Network*, INFOR 28, No. 4 (1990), 363–379.
16. V. Raman, *Finding the best edge-packing for two-terminal reliability is NP-hard*, JCMCC 9 (1991), 91–96.
17. J.T. Robacker, *Minmax theorems on shortest chains and disjoint cuts of a network*, Memo RM-1660-PR, The Rand Corporation (1956).
18. J.G. Shanthikumar, *Bounding Network-Reliability Using Consecutive Minimal Cutsets*, IEEE Transactions on Reliability 37(1) (1988), 45–49.
19. D.R. Shier, "Network Reliability and Algebriac Structures", Oxford Science Publications, Oxford University Press, 1991.
20. L.G. Valiant, *The complexity of computing the permanent*, Theoretical Computer Science 8 (1979), 189–201.
21. D.K. Wagner, *Disjoint (s,t)-cuts in a network*, Networks 20 (1990), 361–371.