**Title**
Min Cuts Without Path Packing



**Authors**
Tucker, Paul A
Hu, T. C.
Shing, M. T.

**Publication Date**
1999-06-17

Peer reviewed

# Min Cuts Without Path Packing

P. A. Tucker* T. C. Hu
M. T. Shing†
*ptucker@cs.ucsd.edu, hu@cs.ucsd.edu, mantak@cs.nps.navy.mil*
UCSD Computer Science and Engineering Dept.
Technical Report CS99-625

June 17, 1999

**Abstract**

This report presents research into a fundamentally new approach to finding all pairwise minimum cuts in a network that can utilize optimality conditions other than those characterized by Mengers theorem or the max-flow min-cut theorem. The focus is on vertex degree domination, rather than construction of saturating paths. We have not been able to show a polynomial-time, deterministic algorithm of this kind, but the investigation has yielded many new insights into the structure of minimum cuts in a network and heuristics for discovering them.

## 1 Introduction

The topic of network flows and minimum cuts has become well established within combinatorial optimization, to the point where it may be considered to encompass an invaluable body of fundamental techniques second perhaps only to linear programming and its related techniques in terms of the number and practical importance of its applications. Prominent applications include efficient communications, distribution, location, production, routing, and scheduling problems with instances in almost any large-scale enterprise.

Combinatorial problems of practical interest generally fall into two classes: those that are computationally tractable, *i.e.* have known polynomial-time solution algorithms, and those that are not. Once a problem is discovered to be tractable, a good algorithm is often discovered quickly and then little further progress is made even when no tight lower bound is known. Among tractable combinatorial problems, network flows and cuts are unusual for having yielded a steady progression of improved solutions by new techniques over several decades. Like linear programs, flow and cut problems are characterized by a primal-dual relationship that leads to efficient solution techniques. The vertex version of Mengers

*Supported in part by a National Science Foundation Graduate Fellowship.
†Computer Science Dept. Naval Postgraduate School, Monterey, CA 93943

theorem [24](1927) proved a strong duality between the problem of finding a maximal number of vertex-disjoint paths between a given pair of vertices, and the minimum number of vertices whose removal leaves that pair in unconnected components. Duality between a maximal packing of edge disjoint paths and a minimum set of edges whose removal separates the pair of vertices is the edge version of this theorem. Until fairly recently, all algorithms for finding a minimum cut relied essentially on solving the dual problem of maximum flow, so the history of minimum cut algorithms is largely that of maximum flow algorithms. In the 50's flows were formulated as a generalization of path packing that is applicable to directed, capacitated networks, and the max-flow min-cut theorem [9] was first proven. Perhaps the first programmed, algorithmic method in this family of optimization problems was Dantzig's linear program formulation of a transportation problem [6], which by relying on the simplex method for solution, was not polynomial time bounded in the worst-case. Within the decade, though, special-purpose polynomial time algorithms based on the augmenting path method were known. Since then a number of small but steady improvements to augmenting path algorithms have been introduced, including the ideas of blocking flows, preflows, and adaptive length functions.

Although the maximum flow and minimum cut problems are duals, they are not exactly equivalent, and their applications do differ. Given the value of a maximum flow, the value of the minimum cut follows, and it is fairly easy to construct a minimum cut from a maximum flow. However, it is not easy to construct a maximum flow just from a minimum cut.

In the last decade, as max flow techniques have matured, several alternative approaches have become known that represent improvements for some applications that need to find cuts quickly, without needing flows. Gabow [10], building on a characterization of the problem by Edmonds [8], which is more closely related to the edge version of Mengers theorem than to the max-flow min-cut theorem, introduced a method of finding the global minimum cut (the one smallest valued cut separating any two vertices) by constructing a maximum directed tree packing. Nagamochi and Ibaraki [27] introduced a novel and efficient method for constructing maximal undirected spanning forests from which followed a new method for finding the global minimum cut. Karger [21] introduced a randomized algorithm that finds the global minimum cut with high probability without constructing any kind of packing at all. Karger's technique yields a global minimum cut while giving very little assistance to the discovery of an actual maximum flow crossing the cut. Moreover, its computational complexity is less than that of any flow-based technique, suggesting that the min cut problem is fundamentally easier than max flow, and does not require a packing approach.

This report is the result of research into possible techniques for finding minimum cuts without constructing maximum flows, or even any kind of path packing. In the broadest scope, we attempted to find a new characterization for minimum cuts, leading to an efficient algorithm, that does not rely on any duality between cuts and path packings. We drew instead upon a characterization of minimum cuts following from the Gomory-Hu result of 1961 [16] regarding the inter-relationships of all minimum cuts in a network. Rather than identifying cuts individually, by the dual problem of max flow, we have sought to find all fundamental min cuts in a network "simultaneously" by the necessary properties they collectively satisfy in partitioning the network. Our main contributions include (1) new problem formulations for finding min cuts without max flows, (2) necessary and sufficient

conditions characterizing their optimal solution, (3) a variety of new sufficient conditions and heuristics for finding minimum cuts, (4) the concept of self-dominating sets, their relation to minimum cuts, and an algorithm for finding them and (5) a new randomized algorithm for finding the Gomory-Hu cut tree directly.

# 2 Related Work

The minimum network cut problem has been very well studied. Its history is mostly that of maximum flows. It is not appropriate to include an extensive overview of prior research on that topic here (see Ahuja, Magnanti and Orlin [2] for a recent, comprehensive text on network flow topics). However, we will summarize here some of the significant results that influenced the direction of our research effort.

The first major result in network flows and cuts as a topic in combinatorial optimization was the max-flow min-cut theorem proven by Ford and Fulkerson [9] (also credited to Kotzig [23]) which states that the maximum flow between two vertices in a network is exactly equal to the minimum cut separating them. This theorem is a generalization to directed, capacitated networks of the basic duality for graphs first identified by Mengers theorem [24]. Since then and until quite recently, all minimum cut algorithms have relied on this duality to find minimum cuts by constructing a saturating flow.

A network flow is an assignment of value and direction to network edges in a way that meets the basic flow feasibility requirements, *i.e.* the capacity of no edge is exceeded by its flow value, and the net flow into every vertex (after flow out is subtracted) is zero, except for a *source*, from which all flow originates, and a *sink* into which all flow terminates. The flow concept is thus dependent on identification of a source and a sink vertex. It is easy to create an initial feasible flow from the source to the sink by finding any path connecting them. Such a path can be found by breadth-first search, for example. Increasing the flow to maximum can be achieved by iteratively finding augmenting paths from source to sink, *i.e.* paths consisting of edges with unused capacity, or capacity that can be freed by feasibly altering the existing flow. When no augmenting path remains, there must exist a collection of flow saturated edge that partitions the network in two, and the current flow is provably maximum. The augmenting path search component of a maximum flow algorithm can be designed so that a minimum cut separating the source and sink vertices is easily identified from the saturated edges when no remaining augmenting path can be found.

Improvement in maximum flow algorithms has come mainly from techniques that discover augmenting paths in a way that minimizes the number of augmentations necessary to saturate the minimum cut. Landmark results in this effort include blocking flows [7], preflows [15] and adaptive length functions [14]. Current best maximum flow algorithms have become quite sophisticated, utilizing combinations of techniques, including sparsification, to achieve their worst-case bounds (see Goldberg [13] for a survey).

Although a flow is not explicitly a collection of disjoint (capacitated) paths, it is simple to decompose it into one, so we can think of a maximum flow as a maximum packing of paths. Only in the past decade have non-flow and non-path-packing techniques been introduced for finding minimum cuts.

The first algorithm for finding the global minimum cut in a network was the Gomory-Hu algorithm [16] that utilized an $s$-$t$ min cut algorithm as a subroutine, up to $n-1$ times. In 1990 Padberg and Rinaldi [29] published a report on a practical algorithm for finding the global minimum cut that, while still max-flow based, was novel for incorporating a number of heuristics that identified lower bounds on some cuts in circumstances that allowed far fewer than $n-1$ calls of the max-flow subroutine, in typical problem instances. A more recent and comprehensive study [5] of minimum cut algorithm implementations demonstrated the effectiveness of some of these heuristics for improving the performance of almost all algorithms studied.

Gabow [10] introduced a method that does not utilize maximum flows, as such, but does rely on an analogous concept, that of a maximum packing of arborescences. Gabow's algorithm is based on a theorem of Edmonds [8] which states that the minimum cut separating a vertex $s$ from any other vertex in the network is equal to the maximum number of disjoint spanning trees rooted in $s$. (As stated, this theorem and the algorithm apply only to unweighted graphs. Gabow and Manu [11] have extended the algorithm to capacitated networks.) By using tree packings, Gabow's algorithm finds the minimum cut separating $s$ from some part of the network by saturating it with simultaneous paths to every vertex $t$ on the other side of that cut.

Nagamochi and Ibaraki [27, 28] introduced a different kind of exhaustive tree constuction algorithm that has proven useful in two distinguishable ways for finding minimum cuts. Their algorithm for an undirected multigraph partitions the edges $E$ into a series $E_1, E_2, \ldots, E_k$ of maximal spanning forests, where $E_i$ induces an acyclic subgraph that connects every connected component of $G' = (V, E - (E_1 \cup E_2 \cup \ldots \cup E_{i-1}))$. In the process, it assigns a total ordering over the vertices, based on their adjacency to the previously ordered set. For this reason their algorithm, especially its generalization to capacitated networks, is also known as *maximum adjacency* indexing. A partitioning into maximal spanning forests differs significantly from a maximum packing of spanning trees in that there is no guarantee that a given forest does not cross a given minimum cut multiple times, so that the first $i$ forests together may saturate a cut of value greater than $i$, even the global minimum cut (the least cut that partitions $V$ into two non-empty sets). However, this kind of forest partitioning can be done very quickly, in $O(m + n \log n)$ time, and is thus useful as a technique for sparsification. A theorem says that if the minimum cut between vertices $s, t$ in $G = (V, E)$ is $\geq i$, then the minimum cut between them in $G' = (V, E_1 \cup E_2 \cup \ldots \cup E_i)$ is still $\geq i$. Hence, if we seek a minimum cut or maximum flow known to have some upper bound, sparsifying the graph first by getting rid of edges in extraneous spanning forests can reduce the time needed to find it. A second theorem leading directly to a new minimum cut algorithm is that the minimum cut between the last two vertices indexed by maximum adjacecy is exactly equal to the degree of the last. In other words, maximum adjacency indexing tells us the value of the minimum cut between these two vertices, even though it does not appear to help find a saturating flow between them (except by sparsification, as mentioned). The global minimum cut can thus be found by $n-1$ iterations of the maximum adjacency indexing algorithm, each followed by contraction of the last two vertices. Although maximum adjacency indexing does not explicitly reveal a maximal collection of disjoint paths between the last two vertices, the proof of their minimum cut value essentially says that the forest partitioning implies that

such a collection exists, hence one might consider it as still in the class of path packing algorithms.

Karger [18, 22] has introduced the first complete minimum cut algorithms that do not involve path or tree construction at all—nor max-flow min cut duality—but instead are based on random sampling. Rather than searching for the minimum cut between a particular pair of vertices, such as is discovered by building one maximum flow, these algorithms target the global minimum cut.[1] The key concept is that this minimum cut is distinguished by being crossed by fewer edges than average, so that if we select an edge at random, the expectation of it crossing the global minimum cut is low. Hence, if we contract a random edge $(u, v)$, we can expect that the minimum cut of the resulting network is the same as that of the original. After $n - 2$ contractions, there is only one cut remaining, whose value and membership are immediate. In any one run of $n - 2$ random edge contractions the chance that the last remaining cut is the global minimum is low, so it is necessary to run the algorithm repeatedly to raise the expectation that it is discovered in some run. However, by strategic allocation of retrials where they help the most (and use of auxilliary techniques like sparsification), the expectation of discovering or approximating it within a relatively small number of runs can be raised quite high.

# 3 Basics

In this section we present definitions of terminology used, together with some basic results relevant to discovery of GH cut trees. Where results proven here are known to have been previously discovered and published elsewhere, it is so noted. Other results are not known to have appeared before.

## 3.1 Terminology

Our terminology is largely conventional. A graph $G = (V, E)$ consists of a set of vertices $V$ and a set of edges $E \subseteq V \times V$. Vertices are denoted by lower case letters, in the style $v_1, v_i, v_j$, or just $u, v$ etc. Sets of vertices are denoted by capital letters, *e.g.* $X, Y$. Where no confusion should arise, we allow $v_i$ to denote the set $\{v_i\}$. For two sets, we use "$-$" to indicate exclusion of common members, so $X - Y$ denotes all members of $X$ that are not also members of $Y$. Consequently, if $X \cap Y = \emptyset$, then $X - Y = X$.

A *network* is a graph with a capacity function $c : V \times V \to R^+$ over its edges. Unless otherwise noted, we consider only *undirected* networks with non-negative real edge capacities. For simplicity of notation, we allow the edge between vertices $u, v$ to be written either $(u, v)$ or $(v, u)$ but only count it as a single edge for measuring the size of $E$. Consequently, $c(u, v) = c(v, u)$. We make the usual assumption that $|V| = n$ and $|E| = m$ for notations of complexity. The capacity function for edges is extended to pairs of vertex subsets, in a

[1] Karger and co-authors have also introduced algorithms for finding *s-t* cuts, based on random sampling, but these utilize flow or path packing methods on a sparisified network. See, *e.g.* Benczúr and Karger [4].

natural way: if $|X| \geq |Y|$ then

$$c(X,Y) \equiv \sum_{u \in (X-Y), v \in Y} c(u,v).$$

Accordingly, in the special case of a single vertex $v \in X$, $c(X,v) = c(X-v,v)$.

A network *cut* is any partition of its vertices into two non-emtpy, disjoint and exhaustive subsets. Cuts can be identified by the set of edges between the two vertex subsets, or by one of those subsets. We choose the latter. A cut may be identified by either of its two defining vertex subsets: if $X \subset V$ identifies a cut, then $\overline{X} = V - X$ is the complementary set that identifies the same cut. The *value* of such a cut is the total weight of edges that cross the cut, *i.e.* connect a vertex in one subset with a vertex in the other. We denote the value of cut $X$ by

$$[X] \equiv \sum_{u \in X, v \in \overline{X}} c(u,v).$$

Observe that necessarily $[X] = [\overline{X}]$.

A cut can also be referred to in cases where we do not know the exact membership of either subset that defines the cut. $C(X,Y)$ denotes *any* arbitrary cut separating the necessarily disjoint vertex sets $X, Y$, and $C^*(X,Y)$ denotes the minimum among such cuts. For example, the minimum cut separating vertices $u, v$ is denoted $C^*(u,v)$. From traditional association with the maximum flow between arbitrary source and sink vertices, such a min cut between two specific vertices is usually referred to as an *s-t min cut*.

In contrast, the *global minimum cut* is the least cut in the network. The notation $\lambda(X)$ indicates the value of the least cut partitioning a non-empty subset of vertices:

$$\lambda(X) \equiv \min_{Y \subset X, |Y|>0} c(Y, X-Y).$$

We say that this function gives the *inner strength* of the subnetwork induced by $X$. Observe that the global minimum cut of the network $G$ has value $\lambda(V)$.

For simplicity, we allow lexical adjacency to abreviate union of set parameters of these various functions. For example,

$$c(XY, ABu) \equiv c(X \cup Y, A \cup B \cup \{u\}).$$

A flow in an edge is a function $f : V \times V \to R$ modeling actual channel utilization for shipment. Flows are easier to reason about in a directed network, which can be obtained from an undirected network by treating each edge as two arcs of identical capacity but opposite direction. A feasible flow must obey two general constraints

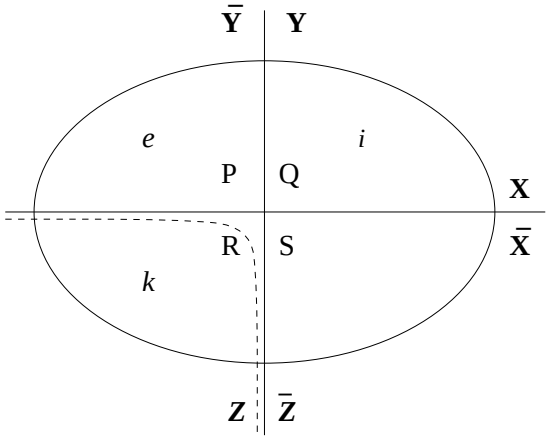
$$f(u,v) \leq c(u,v)$$

and

$$\forall v \neq s,t \quad \sum f(u,v) - \sum f(v,u) = 0$$

specifying that the flow in no arc exceeds its capacity, and all flow is conserved everywhere except the designated source and sink vertices. Without specifying its value, the maximum flow possible from vertex $s$ to vertex $t$ is denoted $f^*(s,t)$.

Figure 1: Non-crossing minimum cut




## 3.2 Some Classical Results

The main classical result in this area is the max-flow min-cut theorem [9], which asserts that the minimum cut separating a source and sink vertex in any network has exactly the same value as the maximum flow from source to sink. It proves that the two problems are duals: a demonstration of any feasible flow imposes a lower bound on the minimum cut, while any cut imposes an upper bound on the maximum flow. As duals, a pair of matching solutions provides a *certificate of optimality* for both: a demonstration of a flow that saturates a cut separating its source and sink suffices to prove optimality of each solution.

Another classical result, due to Gomory and Hu [16], shows that all $s$-$t$ minimum cuts in a network can be characterized by only $n-1$ *fundamental cuts*. (There are $\binom{n}{2}$ choices of $s,t$ from $V$, and there may be more than one cut with the minimum value for many of these pairs; the actual number of distinct $s$-$t$ minimum cuts may be exponential in $n$.) Gomory and Hu showed that there are exactly $n-1$ *non-crossing* minimum cuts that partition the network into a tree structure and give optimum values for every $s$-$t$ min cut.

Two cuts $X, Y$ are said to *cross* each other if each of the four sets

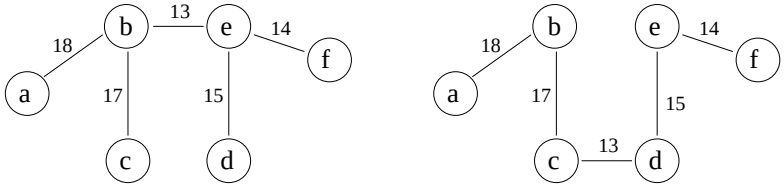
$$X \cap Y,\ X \cap \overline{Y},\ \overline{X} \cap Y,\ \overline{X} \cap \overline{Y}$$

is non-empty. The Gomory Hu result follows from the following theorem regarding non-crossing of minimum cuts.

**Theorem 1 (Gomory-Hu)** *Consider any minimum cut $Y = C^*(i,j)$ separating two arbitrary vertices $i, j$. Assume without loss of generality that $i \in Y$, $j \in \overline{Y}$. For any two vertices $e, k \in \overline{Y}$, if $X = C^*(e,k)$ is a minimum cut separating them, and $X$ crosses $Y$ then there exists another cut $Z$ such that $[Z] \leq [X]$, $k \in Z$, $e \in \overline{Z}$, and $Z$ does not cross $Y$.*

**Proof.** Let the network be divided into quadrants $P, Q, R, S$ by the cuts $X$ and $Y$ as shown in Figure 1. By definition, the value of the cut is equal to the complete capacity of the

Figure 2: Flow equivalent networks




connections between the sets of vertices on either side.

$$c(Y, \overline{Y}) = c(P, Q) + c(P, S) + c(Q, R) + c(R, S) \tag{1}$$

Assume without loss of generality that $i \in Q$, then since $Y$ is the minimum cut separating $i$ and $j$,

$$[Y] \leq c(P, Q) + c(Q, R) + c(Q, S). \tag{2}$$

From equations 1 and 2 it follows that

$$c(R, S) \leq c(Q, S) \tag{3}$$

and hence

$$\begin{aligned} c(P, R) + c(Q, R) + c(R, S) &\leq c(Q, S) + c(P, R) + c(Q, R) + c(P, S) \\ c(R, P \cup Q \cup S) &\leq [X]. \end{aligned}$$

Let $Z = R$ and the proof is complete. ∎

## 3.3 Flow Equivalence and the GH Tree

Two networks defined over the same vertex set $V$ are *flow equivalent* iff the maximum flow between any two vertices is the same in both networks. In consequence of the max-flow min-cut theorem, all pairwise minimum cut values are identical in any two flow-equivalent networks, but the actual minimum cuts may differ. Figure 2 shows two flow equivalent networks in which $C^*(d, e) = \{d\}$ on the left, but $C^*(d, e) = \{e, f\}$ on the right.

The Gomory-Hu all-pairs minimum cut tree (hereafter referred to as the *GH tree*) of a network is any tree that is flow equivalent to the network and has the additional property that the weight of every tree link is equal to the value of the network cut identified by the subset of vertices on either side of the tree link. (For clarity, we usually speak of a network as having vertices and edges and its GH tree as having nodes and links, though of course the tree nodes are identified one-to-one with the network vertices.)

A consequence of this definition of the GH cut tree is that the minimum network cut between any two vertices is exactly equal to the minimum weight of a link in the GH tree path joining the corresponding nodes. Note that, in general, multiple trees exist which satisfy the GH tree definition. However, for simplicity in discussion we usually prefer the tree of smallest diameter, and pick its internal node of greatest sum of link capacities as the root.

## 3.4 Simple Properties of Tree Edges

Let $G$ be any network, and $T$ be the GH tree of that network. Any network edge that is also a link in the tree is a *tree edge*. Any other network edge is a *non-tree edge*. Any link in the tree that is not an edge in the network is a *virtual tree edge*. The capacity of a tree link is denoted $l(u, v)$, to distinguish it from the capacity of a network edge connecting the corresponding vertices.

**Lemma 2** *If $x$ is a leaf in the GH tree, and $(x, y)$ is a tree link, then*

$$f^*(x, y) = l(x, y) = [x].$$

**Lemma 3** *The degree of any vertex $x$ in an undirected network must be at least as high as the capacity of any of its edges in a flow equivalent network (including the GH tree).*

**Proof.** The amount of flow out of a network vertex is limited by its degree; the flow between any two nodes in a flow-equivalent network is at least as great as the capacity of an edge connecting them; hence the degree of a vertex in the original network must be at least as great as the capacity of any edge adjacent to that node in a flow-equivalent network. ∎

**Lemma 4** *If $x$ is the unique highest-degree vertex of a network, then $x$ cannot be a leaf of the GH tree.*

**Proof.** Let $x$ be the unique highest-degree vertex. assume without loss of generality that the network has more than 2 vertices. Suppose towards a contradiction that $x$ is a leaf in the GH tree; then it must have an edge to some one other node $y$. Since $x$ is a tree leaf, $C[xy] = c(x)$, *i.e.* the flow from $x$ to $y$ is equal to the degree of $x$. Then by the previous lemma $c(y) \geq c(x)$, but that contradicts our assumption that $x$ is the unique highest-degree vertex. ∎

**Lemma 5** *If the GH tree is unique, then changing the weight of any tree edge changes the value of exactly one of the $n-1$ fundamental min cuts, but changing the weight of a non-tree edge changes the weight of two or more min cuts.*

## 3.5 Contraction

An important property of the fundamental cuts encoded by the GH cut tree is that, because they are non-crossing, all of the vertices on one side of any fundamental cut can be contracted into a single vertex, and all of the fundamental cuts on the other side are unchanged in the resulting network.

*Contraction* is the operation of substituting a single vertex for a subset of vertices, while merging multiple edges into a single edge. For $G = (V, E)$ the result of contracting subset $S \subset V$ is $G/S = (V', E')$ where $V' = (V - S) \cup \{v_S\}$ and $v_S$ is a new vertex resulting from the contraction of $S$, and $E'$ contains all edges of $E$ joining vertices in both $V$ and $V'$ and an edge $(u, v_S)$ for every vertex $(u, v) \in E$ where $u \in V'$ and $v \in S$. There is also a new

capcity function $c'$ for the contracted network which yields identical values for edges in both $E$ and $E'$ but the capacity of an edge adjacent to $v_S$ is

$$c'(u, v_S) = \sum_{v \in S} c(u, v).$$

The result of contracting a single edge may be denoted in the style $G/(u, v)$.

**Lemma 6** *If $S \subset V$ contains all and only the vertices in a subtree of the GH tree of the network, then the GH tree of $G/S$ is identical to that of $G$, except that the subtree containing $S$ has been reduced to a leaf node.*

A consequence of this lemma is that if, in a network, we contract the vertices of any GH subtree, the pairwise minimum cut between any two remaining vertices is unaffected. An important special case is that contracting a tree leaf into its parent does not affect any other $s$-$t$ min cut in the network.

## 3.6 Domination

In addition to saturation by a maximum flow, there is one other known way of deterministically identifying minimum cuts: the concept of vertex degree domination. The first well-known algorithm to incorporate domination was described by Padberg and Rinaldi [29]. In the worst case their algorithm still requires approximately $n-1$ $s$-$t$ max flow calculations, but it includes a number of heuristics which enable it recognize some instances in which two vertices cannot be separated by the global minimum cut, and hence can be contracted without affecting that cut, thereby saving a max flow calculation.

The Padberg and Rinaldi domination heuristic is the simplest of the domination conditions, one we call a *dominating edge*. An edge $(x, y)$ is a dominating edge at $y$ iff

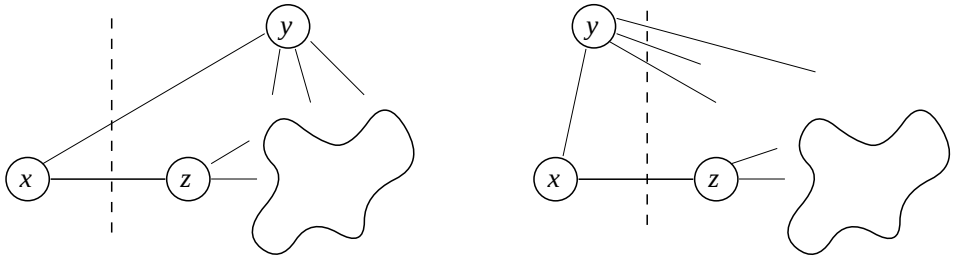
$$c(x, y) > \sum_{v \neq x} c(v, y).$$

A dominating edge is a very special situation, that leads immediately to recognition of a fundamental cut. A vertex may not have a single dominating edge, and if not, it then has multiple dominating sets of edges. A *dominating set* of edges (or neighboring vertices) is any set of edges of a vertex $v$ whose sum of weights is strictly greater than $[v]/2$. [2] The *dominating factor* of a vertex is the size (number of edges) of its smallest dominating set.

**Lemma 7** *If $(x, y)$ is a dominating edge, then $(x, y)$ is a GH tree edge.*

[2] A note on ties: In our definitions of dominating edge and dominating set we specify that the weight of a dominating set be strictly greater than $[v]/2$. If we loosen this to $\geq$ we can still discover fundamental cuts by this concept, but ties (the equality case) admit different possible shapes of the GH tree. By insisting on strict inequality, we favor shallow and bushy GH trees over deep and narrow trees, minimizing the tree diameter. However, in implementation an algorithm may choose non-strict inequality for the performance advantage of more recognized dominating edges.

Figure 3: Dominating edge is a tree edge




**Proof.** Assume without loss of generality that $(x, y)$ is a dominating edge at $y$, and suppose towards a contradiction that $(x, y)$ is not a GH tree edge. In that case there must be some other tree edge $(x, z)$ on the path between $x$ and $y$. Refer to Figure 3 for an illustration. The claim that $(x, z)$ is a tree edge is equivalent to claiming that the cut separating all the nodes on one side of the tree from the other is the minimum cut separating $x$ and $z$ in the network. However, because $(x, y)$ dominates at $y$, we know that $c(x, y) > \sum_{v \neq x} c(v, y)$. Therefore we can decrease the sum of edge capacities crossing the cut by moving $y$ from the right to the left of the cut, as shown in the figure, violating the assumption that the cut was minimal. Therefore $(x, y)$ must be a GH tree edge. ∎

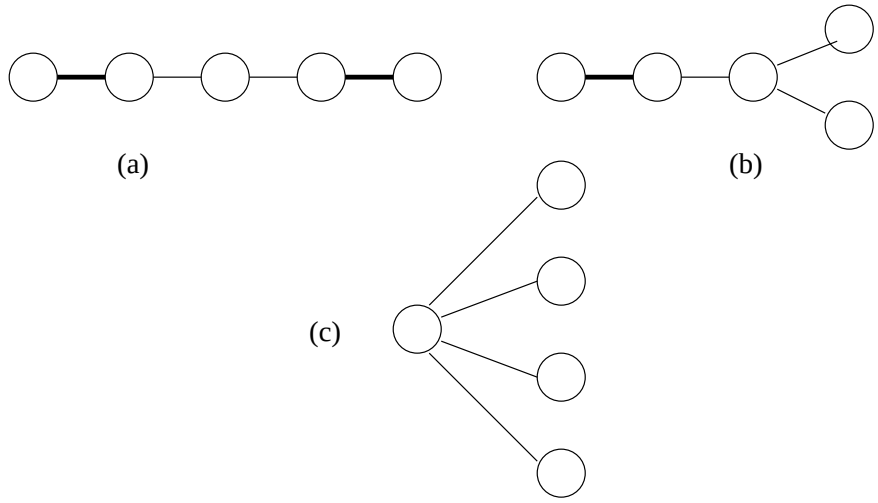
**Lemma 8** *If the dominating factor of every vertex is $n/2$, then the GH tree is a star (has one internal node and $n - 1$ leaves).*

**Proof.** Assume towards a contradiction that there are two internal nodes in the GH tree, $x, y$. Let $X$ be the vertices in the subtree rooted by $x$ that does not contain $y$, and let $Y$ be the vertices in the subtree rooted by $y$ that does not contain $x$. Because these are disjoint subtrees of the GH tree there is a fundamental minimum cut separating $X$ and $Y$. Assume without loss of generality that $|X| \leq |Y|$. Since $x$ is an internal node, there must be some other vertex $z \in X$. By assumption, the dominating factor of $z$ is $n/2$, but since $|X| \leq n/2$, $z$ has fewer than $n/2$ neighbors in $X$ and must have strictly more edge weight adjacent to members of $Y$ than members of $X$. Therefore the cut between the disjoint subtrees rooted by $x$ and $y$ could be decreased by moving $z$ from $x$'s subtree to $y$'s, but that contradicts the original assumption that $x$ and $y$ are two internal vertices in the GH tree. ∎

Single dominating edges and the case of every vertex having a dominating factor of $n/2$ cover the two extreme cases of network topology. If a network is a tree, then every leaf has a dominating edge. When those leaves are contracted with their parents the resulting leaves have dominating edges. Hence, it follows from the dominating edge lemma that any tree network is identical to its GH tree. The other lemma shows that any complete graph, or network whose distribution of edge weights is sufficiently similar to that of a complete graph, has a single star for its GH tree. For small networks, those of 5 vertices or fewer, attention to these cases is sufficient to determine the GH tree without performing any flow calculation.

**Lemma 9** *If $v$ is the only leaf of node $u$ in the GH tree, then $(u, v)$ is a dominating edge at $v$ in the network.*

Figure 4: All five node trees




We will forego proof of this lemma, since it is a simple corollary of Theorem 27 to be proven later.

**Lemma 10** *If $|V| \leq 5$, then either a dominating edge exists, or the GH tree has exactly one internal node.*

**Proof.** We need only show only for $|V| = 5$, since the other cases are simpler. For a 5 node tree, the only possible tree shapes are shown in Figure 4. Both trees (a) and (b) have leaf nodes which are the only children of their parent, and hence by Lemma 9 must have dominating edges. If no dominating edge exists, then (c) is the only possible tree. ∎

## 3.7 Contractable Features

Lemma 6 together with the concept of domination suggests an approach to finding all fundamental pairwise minimum cuts with little or no computation of maximum flows. Domination conditions sometimes allow us to identify a tree edge. If we contract vertices in the subtree on one side of that tree edge, the reduced network is simpler, yet preserves all of the minimum cuts in the un-contracted portion. We may then be able to find another tree edge and another opportunity for contraction, etc.

More generally, there are sets other than GH subtrees that can be contracted for the purpose of simplifying the network and making progress in the analysis of fundamental min cuts.

**Definition 11** *A* strong cluster *is any cut $X$ such that*

$$[X] \leq [Y] \text{ for any non-empty } Y \subset X.$$

Figure 5: Chain network

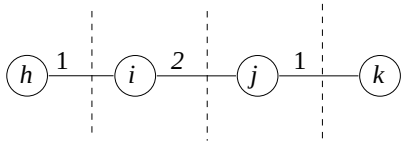
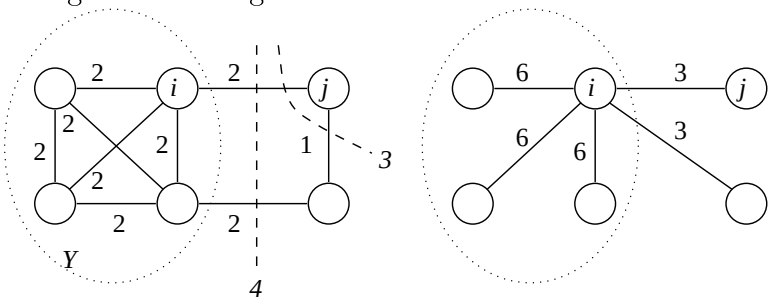



Figure 6: Strong cluster that is not an $s$-$t$ min cut




Another way of stating this is that a strong cluster is any subset of vertices $X$ whose inner strength is not exceeded by its cut value, *i.e.* $[X] \leq \lambda(X)$. Being a strong cluster is independent of being a fundamental minimum cut. However, there are circumstances when we can contract such a cluster, and preserve the structure of some of the fundamental cuts in the rest of the network.

**Lemma 12** *If $X$ is a fundamental minimum cut, then neither $X$ nor $\overline{X}$ is necessarily a strong cluster. If $Y$ is a strong cluster, it is not necessarily a fundamental minimum cut.*

**Proof.** Consider Figure 5. Set $\{h, i\}$ with value 2 is the minimum $s$-$t$ cut $C^*(i, j)$, but $[h] = 1$. Next consider Figure 6. The subset $Y$ forms a strong cluster because each of its members has a capacity 6 connection to the rest of the set, but only a capacity 4 connection to $\overline{Y}$. However, it is not an $s$-$t$ minimum cut. Due to the symmetry, we only need to examine the case for $i, j$, where $C^*(i, j) = 3 < [Y]$.

The righthand part of the figure shows the GH tree for this network. Observe that the strong cluster fails to be a fundamental minimum cut because $Y$ is connected to the rest of the tree by more than one link. ∎

**Lemma 13** *If $X$ and $Y$ are any two* crossing *cuts, then*

$$[Y] \leq [X \cap Y] \Rightarrow [\overline{X} \cap \overline{Y}] \leq [X].$$

**Proof.** Label the quadrants of the network $P, Q, R, S$ as in Figure 1. By assumption

$$c(P, Q) + c(P, S) + c(Q, R) + c(R, S) \quad \leq \quad c(P, Q) + c(Q, R) + c(Q, S)$$

and then,

$$\begin{aligned} c(P,S) + c(R,S) &\leq c(Q,S) \\ c(R,S) &\leq c(P,S) + c(Q,S) \\ c(P,R) + c(Q,R) + c(R,S) &\leq c(P,R) + c(Q,R) + c(P,S) + c(Q,S) \\ [\overline{X} \cap \overline{Y}] &\leq [X]. \end{aligned}$$

∎

**Corollary 14** *If $Y$ is a strong cluster then for any $u, v \in \overline{Y}$ the value of $C^*(u,v)$ is the same in $G$ and in $G/Y$.*

**Proof.** Take any $u, v \in Y$ and suppose that $X = C^*(u,v)$. If $X$ does not cross $Y$, then contracting $Y$ does not affect the min cut between $u$ and $v$. Suppose that $X$ and $Y$ do cross, then $X \cap Y$ is non-empty. Since $Y$ is a strong cluster, $[Y] \leq [X \cap Y]$, and then by Lemma 13 there exists another cut $\overline{X} \cap \overline{Y}$ whose value is no greater than $X$, and does not cross $Y$. Hence contracting $Y$ does not affect the value of the min cut between $u$ and $v$. ∎

# 4 Local Indexing

Define a local indexing algorithm as one that sequentially assigns indices 1 through $n$ to the vertices of a network by always selecting the best candidate from the neighbors of the already indexed set, on the basis of properties which those vertices individually bear to the indexed set, and which never changes the index of a vertex once it has been assigned. A local indexing algorithm is thus a kind of greedy algorithm that establishes a total order over the vertices. Many of the most basic and broadly useful graph algorithms known to computer science are local indexing algorithms, or use one as an essential subroutine. Depth-first search and breadth-first search are both instances, and so are Prim's minimum spanning tree algorithm and Dijkstra's single-source shortest-path algorithm. In some maximum flow algorithms, a local indexing is used to compute a distance function that is then used to find a good augmenting flow.

With respect to finding minimum cuts, probably the most useful known local indexing is the maximum adjacency algorithm. In this section we examine various bounds on connectivity implied by possible outcomes of this algorithm. Alternative indexing criteria to those used by maximum-adjacency are possible, and have been investigated, but are not described here because no more interesting bounds were discovered than those known for maximum adjacency.

## 4.1 Maximum Adjacency

The maximum adjacency indexing technique was introduced in a series of papers by Nagamochi and Ibaraki [26, 27, 28] in which it is defined for unweighted multi-graphs and undirected networks with non-negative real weighted edges. The second case is not significantly more difficult to understand than the first, so we will proceed directly to it. The algorithm

begins by assigning index 1 to an arbitrary vertex, and then iteratively assigns the next index to the unassigned vertex of maximum adjacency to those already indexed, until all have been assigned.

**Definition 15** *Let $v_i$ denote the vertex of index $i$, and $V_i = \{v_1, v_2, \ldots, v_i\}$. Then $v_1, v_2, \ldots, v_n$ are indexed by maximum adjacency if for all $1 < i < j \leq n$*

$$c(V_{i-1}, v_j) \leq c(V_{i-1}, v_i).$$

In cases of ties in adjacency, this definition does not specify a preference for one vertex over another. Nagamochi and Ibaraki showed how to implement a maximum adjacency indexing algorithm efficiently in $O(m + n \log n)$ time. Essentially, a priority queue of unindexed vertices is maintained, sorted by adjacency to the indexed set, and every time we assign the next index to a vertex we update its unassigned neighbors' adjacency and their position in the queue.

For convenience in discussion of maximum adjacency indexing, we generalize with the following definition.

**Definition 16** *Disjoint sets $A, B, \ldots, I$ constitute a* maximum adjacency partition *of $V_i$ if $V$ is indexed by maximum adjacency and*

$$A = \{v_1, \ldots, v_a\}, B = \{v_{a+1}, \ldots, v_b\}, \ldots, I = \{v_{h+1}, \ldots, v_i\}.$$

With respect to finding the global minimum cut, the main result proved by Nagamochi and Ibaraki about a maximum adjacency indexing is that the minimum cut separating the last two vertices indexed is equal to the degree of the last. We present an alternative proof of this theorem similar to that of Stoer and Wagner [30].

**Theorem 17 (Nagamochi-Ibaraki)** *If $V$ is indexed by maximum adjacency, then $C^*(v_{n-1}, v_n) = \{v_n\}$.*

**Proof.** We will show this is true for an arbitrary case, without making any special assumptions, so we claim that it holds true in general.

Suppose that $C^*(v_{n-1}, v_n) = A \cup C \cup E$, where $A, B, C, D, E$ is a maximum adjacency partition of $V_{n-1}$ (the other side of the cut is $B \cup D \cup \{v_n\}$). We can always organize the vertices on each side of a cut into sets that are consecutively indexed according to maximum adjacency; in this case we arbitrarily assume that there are five such sets, plus the last vertex $v_n$. By definition of cut capacity we know that

$$\begin{aligned} c(ACE, BDv_n) &= c(A, BDv_n) + c(B, CE) + c(C, Dv_n) + \\ &\quad c(D, E) + c(E, v_n). \end{aligned} \tag{4}$$

By definition of a maximum adjacency partition we know that

$$c(A, v_{a+1}) \geq c(A, v_{b+1})$$

hence

$$c(A, BDv_n) \geq c(A, v_{b+1}(B - v_{a+1})Dv_n).$$

Substituting into equation 4 above we get

$$\begin{aligned} c(ACE, BDv_n) &\geq c(A, v_{b+1}(B - v_{a+1})Dv_n) + c(B, CE) + \\ &\quad c(C, Dv_n) + c(D, E) + c(E, v_n). \end{aligned} \tag{5}$$

Next observe that

$$c(B, CE) = c(B, v_{b+1}) + c(B, (C - v_{b+1})E)$$

and

$$c(A, v_{b+1}) + c(B, v_{b+1}) = c(AB, v_{b+1})$$

so we can simplify equation 5 to

$$\begin{aligned} c(ACE, BDv_n) &\geq c(AB, v_{b+1}) + c(A, (B - v_{a+1})Dv_n) + \\ &\quad c(B, (C - v_{b+1})E) + c(C, Dv_n) + \\ &\quad c(D, E) + c(E, v_n). \end{aligned} \tag{6}$$

All cuts have non-negative capacity, so eliminating the terms $c(A, (B - v_{a+1})Dv_n)$ and $c(B, (C - v_{b+1})E)$, that we won't need for further simplification, preserves the inequality.

$$\begin{aligned} c(ACE, BDv_n) &\geq c(AB, v_{b+1}) + c(C, Dv_n) + \\ &\quad c(D, E) + c(E, v_n) \end{aligned} \tag{7}$$

Notice that we've now moved the first two vertex sets, $A$ and $B$, that were on opposite sides of the cut onto the same side. We continue by the same technique. Now that we've moved $A, B$ together we know by definition that

$$c(AB, v_{b+1}) \geq c(AB, v_{c+1}).$$

Again, we can separate the term

$$c(C, Dv_n) = c(C, v_{c+1}) + c(C, (D - v_{c+1})v_n)$$

and substitute

$$c(AB, v_{c+1}) + c(C, v_{c+1}) = c(ABC, v_{c+1})$$

yielding the new relation

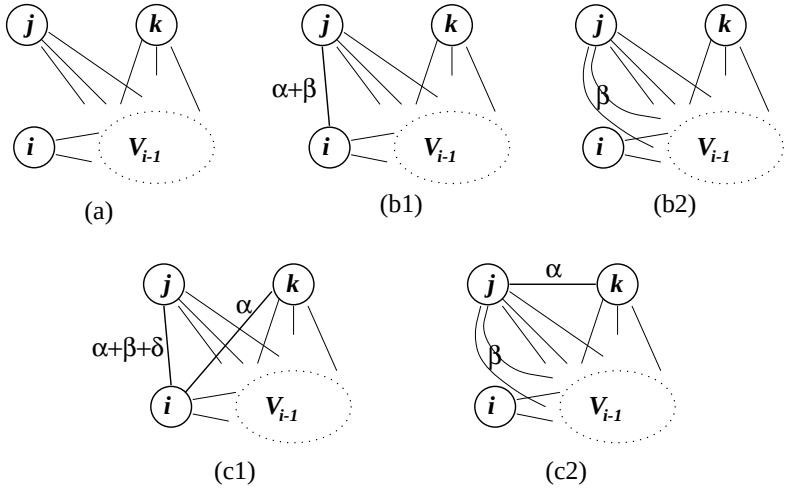
$$\begin{aligned} c(ACE, BDv_n) &\geq c(ABC, v_{c+1}) + c(C, (D - v_{c+1})v_n) + \\ &\quad c(D, E) + c(E, v_n). \end{aligned} \tag{8}$$

Cancelling the non-negative term $c(C, (D - v_{c+1})v_n)$ and repeating the process two more times yields the relation

$$c(ACE, BDv_n) \geq c(ABCDE, v_n) \tag{9}$$

which completes the proof. ∎

Figure 7: Proof cases




In terms of the GH tree, the previous theorem says that a maximum adjacency indexing always assigns the last index to a leaf node.

The next theorem shows how a maximum adjacency indexing establishes lower bounds on flows and cuts. This is an alternative statement and proof of the more general result of Nagamochi and Ibaraki [27], which is the basis of the use of their algorithm for sparsification. The multigraph version of their algorithm applies labels $\{1, 2, 3, \ldots\}$ to edges as the vertices are indexed by maximum adjacency. So long as a vertex remains unindexed, each of its edges receives a new sequential label as soon as the corresponding neighbor is indexed. In consequence, vertex $v_j$ has at least one edge with each label from 1 to the number of neighbors of $v_j$ that were indexed before it. The theorem of Nagamochi and Ibaraki says that all the edges with label $i$ form a maximal spanning tree in the graph obtained by removing all edges of label $< i$. For a network with capacitated edges the labeling is a little more complicated, and it is not always possible to separate the edges into a packing of unit weight spanning trees, but we can show equivalently that the maximum flow between any pair of vertices is lower bounded by the minimum adjacency that either of them bears to any indexed set that does not contain either.

**Theorem 18 (Nagamochi-Ibaraki)** *For any maximum adjacency indexing,*

$$f^*(v_j, v_k) \geq \min\left(c(V_i, v_j),\ c(V_i, v_k)\right) \text{ for } i < j, k. \tag{10}$$

**Proof.** This lemma is easiest to prove by induction on the length of an indexed prefix.

*Base case:* The set $V_1$ consists only of one vertex. If $c(V_1, v_j) > 0$, then there exists an edge $(v_1, v_j)$; likewise for $v_k$. Since both edges meet in the same vertex $v_1$, if they exist at all, $f^*(v_j, v_k)$ is clearly at least the minimum of the two edge capacities.

*Inductive step:* Suppose the lemma is true for all sets $V_1, V_2, \ldots, V_{i-1}$; we will show it is also true for $V_i$. Figure 7 illustrates the relevant cases. Vertex $v_i$ is distinguished because it is not in $V_{i-1}$.

In case (a) neither $(v_i, v_j)$ nor $(v_i, v_k)$ exists, therefore

$$\begin{aligned} c(V_i, v_j) &= c(V_{i-1}, v_j) \\ c(V_i, v_k) &= c(V_{i-1}, v_k) \end{aligned}$$

and the lemma holds by inductive hypothesis.

In the second case shown in (b1) $(v_i, v_j)$ exists but not $(v_i, v_k)$. Assume without loss of generality that $c(v_i, v_j) = \alpha + \beta$ where

$$\beta = \min \begin{cases} c(v_i, v_j) \\ c(V_{i-1}, v_k) - \min(c(V_{i-1}, v_j), c(V_{i-1}, v_k)). \end{cases} \tag{11}$$

Since edge $(v_i, v_k)$ does not exist, it follows that

$$c(V_{i-1}, v_j) + \beta = \min(c(V_i, v_j), c(V_i, v_k)).$$

The lemma thus asserts that

$$f^*(v_j, v_k) \geq c(V_{i-1}, v_j) + \beta. \tag{12}$$

By the inductive hypothesis we already know that the maximum flow is at least the first term of the righthand side; we only need to show that it is at least $\beta$ more. By the definition of a maximum adjacency indexing, since $v_j, v_k \notin V_i$, it must be the case that

$$\begin{aligned} c(V_{i-1}, v_j) + \beta &\leq c(V_{i-1}, v_i) \\ c(V_{i-1}, v_i) &\geq \beta. \end{aligned}$$

Assume without loss of generality that $v_i$ is linked to $V_{i-1}$ by edges $(v_a, v_i), \ldots, (v_c, v_i)$. Then we know we can route $\beta$ worth of flow from $v_j$ through $v_i$ into $V_{i-1}$ by these edges without interfering with any existing flow between $v_j$ and $v_k$. In fact, we can replace $(v_i, v_j)$ with augmenting edges that go directly to the same vertices and whose capacities sum to $\beta$ and obey the restriction $c(v_b, v_j) \leq c(v_b, v_i)$, without altering the maximum adjacency partition of the graph. Therefore, the maximum flow between $v_j$ and $v_k$ passing through $V_i$ is equivalent to the flow passing through the augmented $V_{i-1}$ shown in (b2). Since that augmented graph is a feasible maximum adjacency indexing up through vertex $i - 1$, equation 12 follows by inductive hypothesis.

In the third case, shown in (c1), both edges $(v_i, v_j)$ and $(v_i, v_k)$ have positive weight. Assume without loss of generality that their weights are $\alpha + \beta + \delta$ and $\alpha$, respectively, where $\beta$ is defined as before. Observe that

$$\min(c(V_i, v_j), c(V_i, v_k)) = \alpha + \beta + \min(c(V_{i-1}, v_j), c(V_{i-1}, v_k)). \tag{13}$$

As in the previous case we know that we can augment $V_{i-1}$ to get a valid maximum adjacency indexing through index $i - 1$ where the relative degree of $v_j$ is increased by $\beta$. Hence we only need to show that an additional $\alpha$ worth of flow can be routed from $v_j$ to $v_k$ without interfering with existing flows and exceeding any edge capacities in $V_i$. And that is obviously possible using the path $v_j, v_i, v_k$ that does not contain any edges in or linked to $V_{i-1}$. Hence

$$f^*(v_j, v_k) \geq \alpha + \beta + \min(c(V_{i-1}, v_j), c(V_{i-1}, v_k)) \tag{14}$$

and the lemma holds for this case. The intuitive idea of how the extra flow is routed is shown in (c2). The other possible cases are symmetrical to cases (b) and (c), so the lemma is proven. ∎

## 4.2 Extensions

Given the speed with which a maximum adjacency indexing can be computed, it would be extremely convenient if we could get more information out of it, either in a single invocation, or perhaps from several tries, starting with different initial vertices. If would could find at least one tree edge, for example, it would allow the GH tree and all fundamental cuts to be computed faster than any known algorithm. Unfortunately, we have not been able to find a way of using maximum adjacency to discover essential aspects of the GH tree structure. We have counterexamples demonstrating that all of the following conjectures, for example, are *false*.

1. There is a tree edge between $v_n$ and $v_{n-1}$.
2. By varying the starting point (*i.e.* the vertex assigned index 1), it is possible for any leaf node to be indexed last. *(In fact, it is possible that as few as two nodes can receive the last index, no matter which node is indexed first.)*
3. If $A, B, C, D$ is a maximum adjacency indexing, and $|B| > |D|$, then
$$c(AC, BD) \geq c(ABC, D).$$
4. Vertex $v_{n-1}$ is also a leaf node.
5. A maximum adjacency indexing respects the global minimum cut (or any bottleneck cut), *i.e.* does not cross it until all vertices on one side have been indexed.

However, we have been able to prove a few new facts based on watching the minimum and maximum values of certain adjacency measures during the construction of a maximum adjacency indexing.

**Definition 19** *If vertices $V$ are indexed by maximum adjacency, then $\alpha_i \equiv c(V_{i-1}, v_i)$.*

We say that a maximum adjacency indexing (prefix) is *monotonically increasing* if the adjacencies of the vertices at the time they are indexed forms a monotonically increasing sequence, *i.e.* $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_i$. The next lemma says that during construction of a monotonically increasing maximum adjacency indexing, each time that the new vertex indexed is on the opposite side of an arbitrary cut from the last vertex, the minimum value of that cut must be at least the adjacency of the new vertex.

**Lemma 20** *If $V_k$ is a monotonically increasing maximum adjacency indexing, and $Y$ is any cut such that $v_k \in Y$ and $v_{k-1} \in \overline{Y}$, then*

$$[Y] \geq \alpha_k. \tag{15}$$

Figure 8: Lower bound on cut crossings – schema

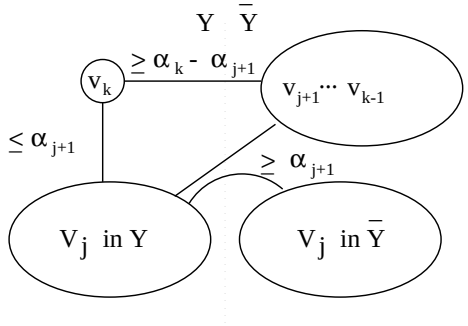
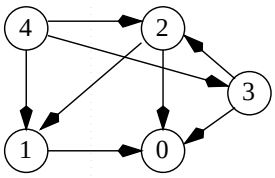



Figure 9: Lower bound on cut crossings – example




**Proof.** We perform induction on the size of $V_k$.

*Base case:* For $V_1$, the only possible cut separating $v_0$ and $v_1$ has the value $\alpha_1$.

*Inductive step:* Assume the lemma holds for $V_1, \ldots, V_{k-1}$. Let $v_j$ be the vertex of greatest index, except for $v_k$, in $Y$. By the inductive hypothesis we know that $[Y] \geq \alpha_{j+1}$. Refer to Figure 8 for an abstract illustration. Since adjacency is monotonically increasing, we know that $\alpha_k \geq \alpha_{k-1} \geq \cdots \geq \alpha_{j+1}$. Since the indexing is done by maximum adjacency we also know that $\alpha_{j+1} \geq c(V_j, v_k)$. By assumption $v_{j+1}, \ldots, v_{k-1} \in \overline{Y}$, so $C[Y, v_k] \leq c(V_j, v_k) \leq \alpha_{j+1}$. Therefore, if $\alpha_k > \alpha_{j+1}$, then at least $\alpha_k - \alpha_{j+1}$ edges must cross the cut from $v_k$ to vertices in $\overline{Y}$, and $[Y] \geq \alpha_k - \alpha_{j+1} + \alpha_{j+1}$, meaning that the lemma holds. ∎

For a simple concrete example, refer to Figure 9. In this example, arrowheads on the edges indicate the adjacency of a vertex to the indexed set at the time it is first indexed (edges are still undirected).

### 4.2.1 Graphs

Maximum adjacency indexing can imply some stronger properties in the special case of $0, 1$ graphs than in capacitated networks. The adjacency of an unindexed vertex can only go up slowly, by adjacency to many indexed vertices, in contrast to the capacitated case where a vertex with a single edge to the most recently indexed vertex can have maximum adjacency by virtue of that one edge having a large capacity. Because adjacency to the indexed set can only increase by 1, each time a vertex is indexed, it is possible to draw stronger conclusions

about the inner strength of indexing prefix sets.

The following lemmas indicate how a monotone-increasing maximum adjacency indexing, in a graph, tends to indicate a region of significant internal strength. Unfortunately, they fall a little short of being able to prove that such a region is a strong cluster.

**Lemma 21** *In a maximum adjacency indexing of a graph, for all $i, j$*

$$c(V_i, v_j) \leq c(V_{i-1}, v_j) + i.$$

**Lemma 22** *If $V_k$ is a monotonically increasing maximum adjacency indexing of a graph, and $Y$ is any cut through it such that $v_k \in Y$ and $v_{k-1} \in \overline{Y}$, then*

$$\min_{v \in \overline{Y}} c(V_k, v) - 1 \leq [Y].$$

**Proof.** Consider that the minimum internal degree in $\overline{Y}$ can be no greater than the internal degree of any arbitrary vertex in $\overline{Y}$, such as $v_{k-1}$. By lemma 20 we know that the cut must be at least the adjacency of the $k$th vertex

$$\alpha_k \leq [Y].$$

By lemma 21 we know that

$$c(V_k, v_{k-1}) - 1 \leq \alpha_{k-1}.$$

By definition

$$\alpha_{k-1} \leq \alpha_k$$

so therefore

$$c(V_k, v_{k-1}) - 1 \leq [Y].$$

∎

**Lemma 23** *The edge connectivity of any monotonically increasing maximum adjacency indexing in a graph is no less than the minimum internal degree of any of its vertices, minus 1.*

$$\min_{i \leq k} c(V_k, v_i) - 1 \leq \lambda(V_k) \tag{16}$$

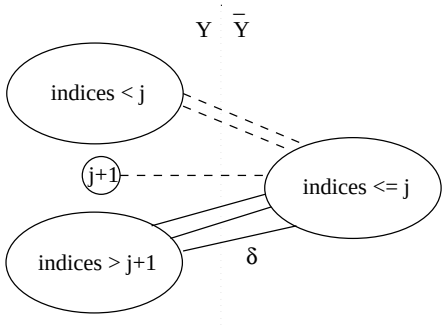
**Proof.** Consider an arbitrary cut $Y$ through the set $V_k$. Assume without loss of generality that $v_k \in Y$ and $v_j$ is the vertex of greatest index in $\overline{Y}$. For an illustration, please refer to Figure 10. By lemma 22,

$$[Y] \geq \min_{v \in \overline{Y}} c(V_{j+1}, v) - 1$$

*i.e.* the cut must have been at least the minimum degree of any vertex in $\overline{Y}$, minus 1, before vertices $v_{j+2}, \ldots, v_k$ were added. This component of the cut value is illustrated with dashed lines in the figure. Let

$$\delta = C[\overline{Y}, v_{j+2}] + \cdots + C[\overline{Y}, v_k].$$

Figure 10: Lower bound on $\lambda$ – schema




This component is illustrated with solid lines in the figure. Notice that $[Y]$ is increased by $\delta$ as the remaining vertices are added, while the minimum degree in $\overline{Y}$ is raised by no more than $\delta$, since the minimum degree cannot be raised by more than the number of edges crossing over into $\overline{Y}$. That is,

$$\min_{v \in \overline{Y}} c(V_{j+1}, v) + \delta \geq \min_{v \in \overline{Y}} c(V_k, v)$$

and

$$[Y] \geq \min_{v \in \overline{Y}} c(V_{j+1}, v) - 1 + \delta.$$

Therefore

$$[Y] \geq \min_{v \in \overline{Y}} c(V_k, v) - 1.$$

∎

Since the minimum internal degree of $V_k$ is also an upper bound on $\lambda$, the above lemma places edge connectivity within a range of 2, for any monotonically increasing maximum adjacency indexing. So although it does not prove that such indexings are always clusters, it shows that they are always very close to being clusters, and the proof reveals why they are sometimes not.

## 4.3 Summary

We examined some properties of maximum adjacency indexing in detail, as they relate to identifying regions of high connectivity. We presented alternative or new proofs, in particular for the new concept of monotone increasing indexings. This style of local indexing was not shown to provide great enough lower bounds on local connectivity to identify tree edges or strong clusters (regions that can safely be contracted in the Gomory-Hu algorithm, so as to simplify the remaining problem). Monotone increasing indexings may be of utility in a Padberg-Rinaldi type algorithm to indicate contractable regions whose local connectivity is greater than some already established upper bound on the global minimum cut being sought.

# 5 The Non-Local Principle

In the previous section we examined some properties of maximum adjacency indexings. We saw that a maximum adjacency indexing is always able to find a single leaf of the GH tree, but cannot tell us which node is its parent, hence it does not necessarily find a tree edge. We also showed other ways in which a maximum adjacency indexing can lower bound the connectivity of subgraphs. In addition to the maximum adjacency rule, we have examined alternative rules for selecting the next vertex to be indexed, without being able to find any that is provably superior for determining network connectivity. In particular, we have not been able to show a general algorithm for constructing the entire GH tree by application of only a local indexing.

The non-local principle for minimum cuts expresses the fact that no local function can be always sufficient for identifying edges or complete subtrees of the GH cut tree. It has implications for the kinds of minimum cut algorithms that can be developed.

**Non-local principle:** Let $\mathcal{N} : (V, E) \times 2^V \rightarrow 2^V$ be the local neighborhood function for undirected networks; for any $S \subset V$

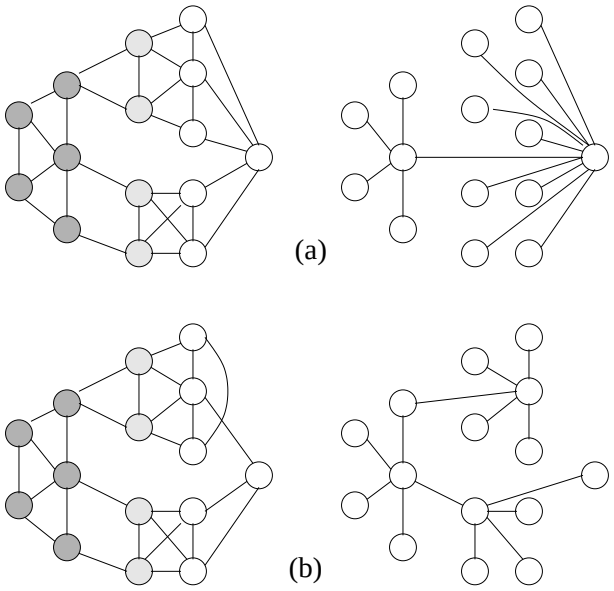
$$\mathcal{N}(G, S) = \{v | v \in S \text{ or } c(v, S) > 0\}.$$

A boolean function $\mathcal{F} : 2^V \times 2^V \rightarrow \{0, 1\}$ is a *local test* for some property if $\mathcal{F}(G/\overline{\mathcal{N}(G,S)}, S) = 1$ if and only if $S$ has the property in question in the original network G. In other words, the function must be able to decide the property when there is no information about entire network outside of the neighborhood of the subset in question, except its total adjacency to each vertex in that neighborhood. Then, *there is no local test for any of the following properties:*

1. *being a GH tree edge*
2. *being an internal node of a GH tree*
3. *being a complete GH subtree*
4. *being a subset that is not partitioned by any min cut between two vertices in its complement.*

The principle can be proven simply by exhibition of two similar networks that are isomorphic under contraction of the complement of the local neighborhood, but which have differing GH tree topologies for the subset in question. Figure 11 shows two $0, 1$ graphs which provide examples of all four non-local properties. The darker shaded vertices show one possible set $S$, and the lighter shaded vertices are the vertices in its local neighborhood. The set $S$ is a complete GH subtree in (a), but not in (b). Similarly, $S$ is not partitioned by any min cut between two vertices in $\overline{\mathcal{N}(G, S)}$ in (a), but in (b) a fundamental cut separating $\overline{\mathcal{N}(G, S)}$ also partitions $S$. Other subsets can be identified that illustrate the other two cases. A weighted network can illustrate similar cases with fewer vertices and no ties in degree.

Figure 11: The non-local principle




## 5.1 Some consequences

It follows immediately that no local indexing algorithm can always identify any of the four properties, when it has only indexed a proper subset of a network's vertices.

Recall that the Gomory-Hu meta-algorithm for finding the cut tree relies upon an $s$-$t$ min cut algorithm to find each tree edge. An $s$-$t$ min cut algorithm divides the vertices into two sets, one containing $s$ and the other $t$, such that each set constitutes a GH cut tree subtree and the two are joined by a single tree edge.

Any $s$-$t$ min cut algorithm, consequently, is thus necessarily non-local in the sense of the principle. It is not always possible to find any subnetwork, on the basis of local tests, that may be ignored (*e.g.* by contracting) when computing an arbitrary $s$-$t$ min cut. Even if $s$ and $t$ are adjacent and of relatively minor degree compared to vertices in other regions, in the worst case an algorithm may have to examine nearly every edge in the network before finding their min cut.

The non-local principle does not establish that local indexing cannot be useful in computing min cuts, just that we cannot always find min cuts locally. Alternative ways of utilizing local indexing that might initially appear of potential utility include computing indexings from every vertex as a starting point, and comparing the results. However, we have not been able to find any algorithm utilizing local indexing that deterministically discovers GH tree edges, and does not essentially compute a saturating path packing.

Note that a preflow seems like a local test, in that it minimally expands its region of partial flow analysis until it discovers whether its starting point is an internal GH tree node, but in cases like those illustrated in Figure 11, its region of partial flow analysis will necessarily include the entire network.

# 6 Alternative Formulations

In order to better study the possibility of discovering minimum cuts without reliance on the max-flow min-cut theorem, we have defined two new problems whose optimal solutions are provably equivalent to finding the GH cut tree, but whose formulations suggest solution methods other than max flow. These formulations apply to undirected networks with non-negative weighted edges.

## 6.1 Minimum Communication Tree

One alternative formulation that has been known for some time is the *minimum communication spanning tree*: Given a network where the weight of each edge corresponds to the number of dedicated wires that must be extended between the two stations modeled by the vertices, find the spanning tree that minimizes total wire length when all dedicated wires are routed along the tree edges, and all tree edges have unit length. Hu [17] proved that the optimal solution is the GH tree of the requirement network, although neither flows nor cuts are needed to define the problem.

## 6.2 Optimum Sequential Contraction

A new problem formulation: *Let the cost of contracting edge* $(u, v)$ *be* $\min([u], [v])$. *Find the least cost sequence of* $n - 1$ *contractions that reduces the network* $G$ *to a single vertex.*

Equivalence of the optimal solution of this problem to the GH tree can be seen by observing that any sequence of contractions resulting in a single vertex defines a tree. Zero contractions defines a tree consisting of a single node, so every vertex in the original network corresponds to a tree of one node. Suppose that vertices $u, v$ are contracted, where $[u] \leq [v]$. The result corresponds to tree created by adding an edge from the root of the tree corresponding to $u$, to the root of the tree corresponding to $v$, with $u$ as a subtree. After $n - 1$ contractions, a tree with $n - 1$ edges is defined. Under this definition, a weight can be assigned to each tree edge corresponding to the cost of the corresponding contraction. Observe that the tree edge weight then also corresponds to the value of the cut separating the vertices on each side of the edge. So the minimum cost sequence of contractions corresponds to the minimum total weight cut tree, which is the GH tree [16].

This problem formulation suggests the error bound minimization heuristic. In planning a contraction schedule, suppose we decide next to contract vertex $u$ into $v$, where $[u] \leq [v]$. If this is not actually the optimum next contraction, how large an extra cost may we ultimately incur? Observe that the degree of the resulting vertex in $G/(u, v)$ is $[u] + [v] - 2c(u, v)$. In a series of $n-1$ contractions, every edge of the original network is contracted once, but by the contraction cost definition, we may pay the weight of an edge multiple times if that edge is adjacent to one, but not both of the vertices involved in several contractions. Taking only a very local view, if we contract $u$ into $v$, we will never be charged for for $(u, v)$ again, but may be charged again for each of $u$'s other edges at least one more time. However, in some alternative contraction schedule, where each of $u$'s neighbors contracted into $u$, those edges would each be charged for one time only. So, the extra-optimal cost of contracting $(u, v)$ is

upper bounded by $[u] - c(u, v)$.

**Definition 24 (Error bound heuristic)** *In scheduling a sequence of contractions, always schedule next the contraction that minimizes* $[u] - c(u, v)$.

This heuristic does not always yield optimum solutions (as implied by the non-local principle), but makes sensible decisions based on a very local view. For example, it always contracts dominating edges, where they exist.

## 6.3 Optimum Circle Partition

Another new problem formulation: Rather than characterizing the fundamental pairwise min cuts by tree edges, we focus on the partition of vertices into nested subsets, by those cuts. Intuitively, imagine drawing $n - 1$ circles around portions of a GH tree so that each circle crosses exactly one tree edge, no two circles cross the same edge, and no circles intersect without proper containment. Then, each circle encloses a unique and complete subtree. The vertices inside each circle are the members of each subset in the partition.

**Definition 25** *A legal circle partition is any assignment of the* $n$ *vertices of a network into* $n$ *subsets of* $V$ *called circles meeting the following restrictions.*

1. *Every circle is uniquely associated with one vertex, which the circle contains, and which is called its* host.
2. *If* $X, Y$ *are different two circles, such that* $|Y| \leq |X|$, *then either* $Y \subset X$ *or* $Y \cap X = \emptyset$.

From this definition it follows that a legal circle partition corresponds to a tree, where every circle contains a complete subtree, and the root of the subtree is that circle's host. There is one "global" circle containing the whole network, for completeness. The $n - 1$ other circles correspond to cuts. Any circle containing only one vertex is an *atomic* circle, and corresponds to a tree leaf. Any legal circle partition must contain at least two atomic circles. All members of a circle other than the host are themselves contained in subcircles. All subcircles directly contained by a circle are said to be *children* of the host.

If we define the *value* of a circle to be the total weight of network edges passing from vertices inside a circle to vertices outside of it, then the *optimum circle partition* is the partition with minimum total sum of circle values. That an optimum circle partition of a network's vertices yields the GH tree of that network follows from the result of Gomory and Hu [16] that the GH tree encodes the $n - 1$ partitioning cuts of least total value.

The optimum circle partition formulation yields some interesting theorems. The first characterizes optimality of a circle partition in terms of local properties, with no reference to maximum flows.

**Definition 26** *Local optimality conditions for a circle partition:*

1. **Host Feasibility***: The degree of the host of the circle must be strictly greater than the value of the circle.*

2. **Internal Adjacency**: *For any non-atomic circle $X$ with host vertex $x$, for every set $S$ that is a union of children of $x$,*

$$\frac{c(S, X - S)}{[S]} > 1/2.$$

3. **External Adjacency**: *Let $y$ be the host of the circle directly containing circle $X$ (i.e. $X$ is the child of $y$). For every set $S$ that is a union of children of $y$ such that $S \cap X = \emptyset$,*

$$\frac{c(S, X)}{[S]} \leq 1/2.$$

4. **Non-Invertability**: *For any non-atomic circle $X$ with host vertex $x$, for every child $Y$ of $x$,*

$$[Y] \leq [X - Y].$$

**Theorem 27 (Necessity)** *The local optimality conditions must hold at every applicable circle in an optimum partition.*

**Proof.** If a simple transformation exists that lowers the sum of values of some set of circles, and leaves the values of all other circles unchanged, then the original circle partition cannot have been optimal. For any violation of any of the above conditions, we will show that such a transformation exists.

*Host Feasibility*: Suppose $x$ is the host of $X$, and $[x] < [X]$. If we make $x$ an atom, moving all of its children outside, to become children of $X$'s parent, then only the value of $x$'s circle changes, decreasing by $[X] - [x]$. (If $X$ is the outermost circle, then $[X] = 0$ and the condition holds for any host. If $[X] = [x]$ then the transformation does not change the sum of circle values, but we insist on strict inequality for structural simplicity, so that we prefer flatter partitions to deeper partitions of the same value.)

*Internal Adjacency*: Suppose that for some $S$, the union of children of $x$, host of $X$, $S$ has *adjacency ratio* $< 1/2$ with respect to $X - S$, that is

$$\frac{c(S, X - S)}{[S]} < 1/2.$$

Then

$$c(S, \overline{X}) > c(S, X - S)$$

and we can decrease the value of $X$ by moving $S$ outside, leaving the value of all other circles unchanged. Again, the condition specifies strict inequality just so that we prefer flatter partitions. (Note that if $X$ is the outermost circle, then no such subset $S$ can exist.)

*External Adjacency*: Suppose that some set $S$, the union of siblings of circle $X$, has adjacency ratio greater than $1/2$ with respect to $X$. Then we can decrease the value of $X$, leaving all other circle values unchanged, by moving $S$ inside $X$.

*Non-Invertability*: Suppose that $Y$ is a child of $x$ in $X$ and $Y$ satisfies the internal adjacency condition with respect to $X$, but $[Y] > [X - Y]$. Then we can transform $Y$ into

$Y'$ by making $X' = X - Y$ a child of the host of $Y$. The value of circle $X$ increases in the transformation, but that of $Y$ decreases. Observe that

$$\begin{aligned} [X'] - [X] &= c(Y, X - Y) - c(Y, \overline{X}) \\ [Y'] - [Y] &= c(Y, X - Y) \\ ([Y'] - [Y]) - ([X'] - [X]) &= c(Y, \overline{X}) \\ [Y] > [X'] &\Rightarrow c(Y, \overline{X}) > 0 \end{aligned}$$

so the result is a net decrease in partition value. ∎

**Theorem 28 (Sufficiency)** *If the local optimality conditions hold for every circle where applicable, then the entire partition is optimal.*

**Proof.** By Theorem 27 we know that the conditions must hold everywhere in an optimal partition. The method of this proof will be to show that there is a contradiction in assuming that they can also hold everywhere in a non-optimal partition.

Assume towards a contradiction that every circle in partition $\mathcal{P}$ meets all of the applicable conditions, but $\mathcal{P}$ is not optimal. Let $\mathcal{P}^*$ be an optimal partition. As usual, one circle is associated with every vertex, the one that hosts it. Since there are exactly $n$ circles in both partitions, and they cannot be identical, among all circles that differ in these two partitions, there must be at least one circle $X$ that is *shallowest* in $\mathcal{P}$. In other words, every child of $X$ is identical in $\mathcal{P}^*$, although they are not necessarily children of $x$ in that partition, and not every child of $x$ in $\mathcal{P}^*$ is a child of $x$ in $\mathcal{P}$.

The figure illustrates an arbitrary example of $X^* \in \mathcal{P}^*$ and several $X \in \mathcal{P}$ that show all the essential possibilities. We assume that $X^*$ may lose some children and gain others in the change to $X$. By cases we can show that the assumption that all applicable conditions are met by all circles in both partitions leads to contradiction.

*Case 1:* $X$ contains a child $d$ that was not a child in $X^*$. Since $X$ is the shallowest changed circle in $\mathcal{P}$, we know that $d$ is identical in both partitions. By internal adjacency in $\mathcal{P}$, it must be the case that

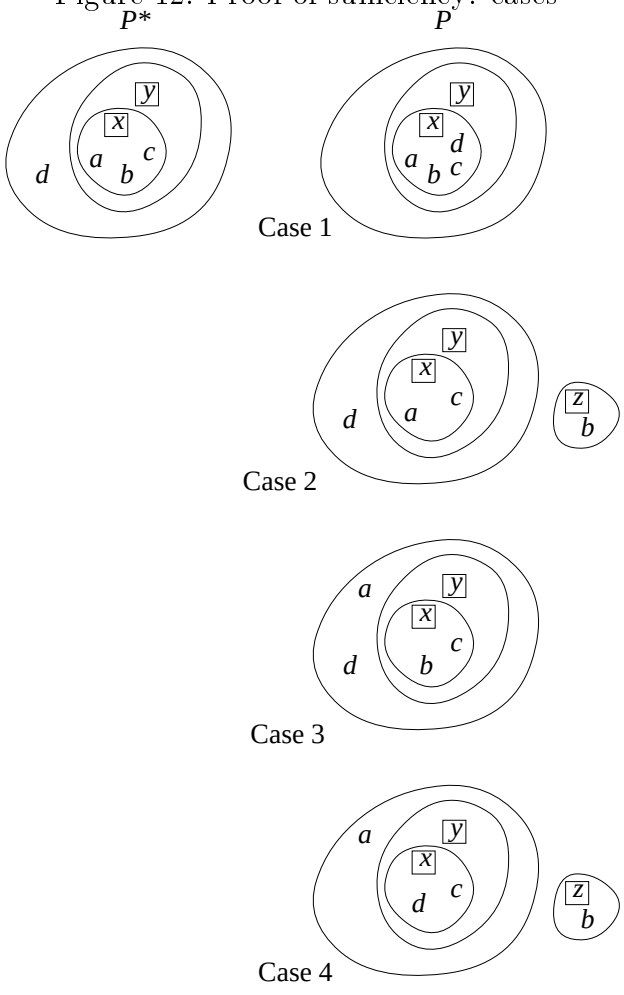
$$\frac{c(d, \{x, c\})}{[d]} > 1/2.$$

But then $\mathcal{P}^*$ must be in violation of external adjacency with respect to $d$ which is outside of circle $X^*$, and may also be outside of one or more circles $Y^*$. It follows immediately that external adjacency is violated if $d$ is a sibling of $X^*$. If an intermediate circle $Y^*$ exists, consider that by assumption $d$ is the same in $\mathcal{P}^*$ and $\mathcal{P}$, so $d$ cannot be an atom hosting the circle containing $Y^*$, and hence it must also violate external adjacency with respect to $Y^*$.

*Case 2:* A child $b$ from $X^*$ is a child in a circle $Z$ not containing $x$. It not possible for $b$ to satisfy internal adjacency of $X^*$ and also $Z$.

*Case 3:* A child $a$ from $X^*$ is a sibling of a circle containing all of $X^* - a$. In other words, $a$ is the outermost child no longer a child of $x$, but not a child in any circle not containing $x$. Clearly, the adjacency ratio of $a$ to $X$ and any intermediate circle $Y$ is $> 1/2$. By assumption

Figure 12: Proof of sufficiency: cases



$a$ has identical structure in $\mathcal{P}$ and $\mathcal{P}^*$, so $a$ cannot be a host. Therefore external adjacency is violated by $a$.

*Case 4:* A combination of two or more of the previous cases. Clearly, $b$ violates internal adjacency, so we can assume that case does not exist. Then $a$ is the outermost child, in which case it violates external adjacency, so we can assume this case does not exist either. Then by the argument above, $d$ cannot be a child in $X$ without causing a contradiction. ∎

## 6.4 Implications of New Formulations

The necessary and sufficient conditions for optimality of a circle partition are local in the sense that for each circle, we only need check properties of combinations of its top level children and siblings. However, in the worst case, that locality extends to the entire network and the number of such combinations is exponential, so theorems 27 and 28 do not obviously lead to an efficient GH tree algorithm. For example, if we begin with a default partition in every vertex is in an atomic circle, except for the one vertex of global largest degree which hosts the circle containing all others, then there are $O(2^n)$ subsets of siblings to be checked in order to test the optimality of each atomic circle. However, if we are able to make a lucky initial guess as to a good partition, then the theorems do lead to efficient checking and correction algorithms for some cases.

**Theorem 29** *Optimality of a circle partition can be checked in $O(n2^B)$ time, where no circle has more than $B$ child circles.*

The parameter $B$ is the maximum branching of the GH tree, *i.e.* the maximum number of children of any internal node. When $B$ is small, the cost of testing optimality of a purported optimal circle partition by checking each of the conditions in Definition 26 is also relatively small.

### 6.4.1 A Correction Algorithm

Within the circle partition formulation, the potentially exponential cost of testing each circle for optimality can be eliminated if we restrict attention to cases of the internal and external adjacency conditions involving only pairs of circles. For example, testing that no single child can improve the value of a circle by being moved outside, or a single sibling by being moved inside. The non-inversion condition applies only to individual child circles, so it is already polynomial time checkable. If we limit consideration to these cases, we can define an algorithm for correcting some near-optimum partitions.

It should be clear that this algorithm can be performed in $O(m)$ time, always terminates, and corrects all single-circle violations of the optimality conditions. Then, if we can devise a heuristic or randomized method that is likely to guess close to an optimal partition, the correction algorithm can be used to improve it, and perhaps converge on the optimal solution.

### 6.4.2 Favoring Arcs

The favoring arc concept derives from the optimum contraction sequence formulation, and is a way of limiting the search space of feasible contractions, suggesting a possible approach

CORRECT-PARTITION:
**for** every shallowest unchecked fundamental circle $X$,from bottom up, **do**
**for** every child circle $Y$ **do**
**if** $Y$ has adjacency to $X$ of $< 1/2$,
push it out to the circle enclosing $X$.
**else if** $Y$ has adjacency $> 1/2$ to any sibling circle $Z$,
push $Y$ into $Z$ and recheck $Y$ within $Z$.
**else if** enclosing $X - Y$ within $Y$ lowers the total partition value,
do so.

to a dynamic programming algorithm.

**Definition 30 (Favoring Arc)** *Vertex $u$* favors *$v$, written $f(u) = v$, if $c(u, v) \geq c(u, w)$ for all $w \neq v$, and $v < w$ for all $w$ such that $c(u, v) = c(u, w)$.*

In other words, each vertex favors the vertex to which it has the single greatest capacity edge.

**Lemma 31** *The favoring function $f$ induces a forest $\mathcal{F}$ of directed arcs over $V$ (the network vertices). Each connected component of this forest contains exactly one pair of vertices that favor each other; otherwise each component is acyclic.*

For network $G = (V, E)$ and any $e \in E$, the favoring forests of $G$ and $G/e$ may be almost identical, or significantly different.

**Theorem 32** *For any fundamental circle $X \subset V$, there is at least one arc in the favoring forest $\mathcal{F}$ that joins two vertices both inside $X$.*

**Proof.** Assume towards a contradiction that for every $u \in X$, $f(u) \notin X$. Then, for any potential host vertex $h \in X$, the connection of this vertex to all the rest of $X$ is less than or equal to $c(X - h, \overline{X})$ (because all of the internal edges to $h$ are $\leq$ to the favoring arcs, which cross the cut). This implies that $[h] \leq [X]$, and $X$ cannot be a fundamental circle. ∎

**Corollary 33** *For any fundamental circle $X \subset V$, there exists a sequence of arcs $e_0, e_1, \ldots, e_k$ belonging respectively to $\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_k$, where $\mathcal{F}_0$ is the favoring forest of $G$ and $\mathcal{F}_i$ is the favoring forest of $G/\{e_0, \ldots, e_i\}$, such that $X$ is contracted to a single vertex in $G/\{e_0, e_1, \ldots, e_k\}$ and no other edges are contracted.*

The preceding corollary says that in searching for the optimum contraction sequence, the search can be restricted to favoring arcs. Although this considerably reduces the search space, it is still exponential, and we have not been able to take advantage of the theorem in order to construct an efficient dynamic-programming algorithm.

# 7 Minimizing $s$-$t$ Min-Cut Calculations

The GH tree can be constructed by the Gomory-Hu algorithm [16] with only $n-1$ $s$-$t$ min cut calculations, many of which are on contracted networks significantly smaller than the original. In this section we consider whether, independent of the $s$-$t$ min cut algorithm employed, that number can be reduced.

Suppose we have an $s$-$t$ min cut oracle and wish to compute the entire GH tree with as few calls to that oracle as possible. We have already shown that whenever a dominating edge exists we can identify a tree edge immediately, and simplify the problem. Similarly, for networks of 5 or fewer vertices the structure of the tree is immediate just by checking for dominating edges. Is there a good way of picking a schedule of $s$-$t$ min cut oracle calls so that the number of small cases and dominating edges encountered is maximized, and hence reliance on the oracle is minimized? There does not appear to be a deterministic way of selecting such a schedule that is any cheaper than calculating the cuts, but here we provide some justification for the following heuristic: *always next calculate the $s$-$t$ min cut between the two vertices of largest degree that have not yet been separated by a cut.*

If we ask the oracle for an arbitrary $s$-$t$ min cut and the size of the smaller cut set is between 2 and 4, its subproblem is no larger than 5 and we can process it without any further calls to the oracle. If it is larger than 4, then each resulting subproblem has no more than $\leq n-4$ vertices. In any of these situations, our ability to solve small networks without use of the oracle implies that overall no more than $n/2$ oracle calls should be necessary. However, if the cut turns out to separate only one vertex from the rest of the network, the remaining subproblem still has $n$ vertices, unless a contraction is feasible. If we index the vertices $v_1, v_2, \ldots, v_n$ by *decreasing* degree, and then ask the oracle for the min cut $C^*(v_1, v_2)$, then by earlier lemmas we know that if this cut is just $\{v_2\}$, then $v_2$ is a leaf node of $v_1$ in the GH tree, and we can contract the two vertices without affecting any other $s$-$t$ min cuts. Contraction may result in formation of dominating edges that can be contracted in turn. This suggests that, unless we can anticipate finding a roughly balanced cut for some other pair (and we know of no way to guess this, except that heuristically two vertices of high degree are more likely to be independent hosts than two vertices of low degree), we should always apply the oracle to the largest two degree vertices. Then, with some luck, even if we find many leaf cuts, we will still find a lot of dominating edges.

Suppose that the degree of each vertex in a network is fixed, but edge capacities are randomly distributed such that, if $w_{ij}$ is a random variable whose value is the capacity of edge $(v_i, v_j)$, then its expectation is

$$E[w_{ij}] = \frac{[v_i] + [v_j]}{2(n-1)}.$$

(Observe that $\sum_{(v_i,v_j)\in E} E[w_{ij}] = \sum_{v\in V} \frac{[v]}{2}$ which is the total capacity of edges.) Then, if vertices are indexed by decreasing degree and we repeatedly call the oracle on the two vertices of largest degree, find the second is a leaf of the first, contract them and continue, then we can expect the $i$th largest degree vertex to be dominated by the contracted super-vertex of

all larger vertices when

$$\begin{aligned} E\left[\sum_{j<i} w_{ij}\right] &\geq E\left[\sum_{i<j} w_{ij}\right] \\ \sum_{j<i} \frac{[v_i]+[v_j]}{2(n-1)} &\geq \sum_{i<j} \frac{[v_i]+[v_j]}{2(n-1)} \end{aligned}$$

which is true for $i > n/2$. In other words, even in this "worst case" where every oracle call finds a single leaf, by always asking for the min cut between the two vertices of largest degree, we can still expect to get about 1/2 of the GH tree edges by dominating edges.

Unfortunately, while the foregoing probabilistic analysis supplies some credibility for the heuristic, it does not really describe the worst case, and we cannot guarantee that any run of the GH cut tree algorithm will find more than a very few dominating edges. The following table shows the adjacency matrix of an example network of 16 vertices in which every vertex but one is a leaf in the GH tree, but the heuristic requires us to call the $s$-$t$ min cut oracle $n-2$ times. Observe that in this example all most all edge capacities are unique, and all vertex degrees are unique, so it is not simply an artifact of permitting ties among edge capacities. (Empty entries are zero.)

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | deg. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 100 | | | | | | | 93 | 9 | | | | | | | 202 |
| 2 | 100 | | 99 | | | | | | | 2 | | | | | | | 201 |
| 3 | | 99 | | 98 | | | | | | 2 | | | | | | | 199 |
| 4 | | | 98 | | 97 | | | | | 2 | | | | | | | 197 |
| 5 | | | | 97 | | 96 | | | | 2 | | | | | | | 195 |
| 6 | | | | | 96 | | 95 | | | 2 | | | | | | | 193 |
| 7 | | | | | | 95 | | 94 | | 2 | | | | | | | 191 |
| 8 | | | | | | | 94 | | | 71 | 24 | | | | | | 189 |
| 9 | 93 | | | | | | | | | | 67 | 27 | | | | | 187 |
| 10 | 9 | 2 | 2 | 2 | 2 | 2 | 2 | 71 | | | | 63 | 30 | | | | 185 |
| 11 | | | | | | | | 24 | 67 | | | | 59 | 33 | | | 183 |
| 12 | | | | | | | | | 27 | 63 | | | | 55 | 36 | | 181 |
| 13 | | | | | | | | | | 30 | 59 | | | | 51 | 39 | 179 |
| 14 | | | | | | | | | | | 33 | 55 | | | 45 | 44 | 177 |
| 15 | | | | | | | | | | | | 36 | 51 | 45 | | 43 | 175 |
| 16 | | | | | | | | | | | | | 39 | 44 | 43 | | 116 |

# 8 Self-Dominating Circles

By Theorem 27 we know that in an optimal circle partition every member of a circle except possibly the host must have an adjacency ratio of $> 1/2$ to the other members of the circle. We have also seen that local indexing algorithms appear insufficient for discovering all fundamental cuts. In this section we investigate a different style of local algorithm that

finds a structure called a self-dominating circle, and then consider how it may in turn be used to find fundamental cuts.

## 8.1 Assumptions and Definitions

The algorithms in this part modify a total ordering over vertices. Since we wish to identify a vertex independent of its index (from 1 to $n$ which induces the ordering), we grant each vertex a *name*, such as "$x$", as necessary. To refer to a vertex by its current position in the ordering, we use the style $v_i$.

We continue the use of "circle" to denote any non-empty set of vertices with a designated host, being the vertex of largest degree (not enclosed in any subcircle). A new concept is that of a *leader*. The leader of a circle is the member vertex of largest index, in the current index ordering.

A *self-dominating* circle (SDC) is any circle each of whose members, except possibly one designated member, is strictly dominated by the other vertices of the circle. We identify an *open* self-dominating circle as one in which the leader is the only vertex that need not be dominated. A *closed* self-dominating circle is one in which we designate some initial vertex as always and necessarily a member, regardless of whether it is dominated, and then require all other vertices (including the leader, if distinct) to be dominated.

It follows from Theorem 27 that a non-atomic fundamental circle is necessarily self-dominating (when its host is excused from the domination requirement), but not all self-dominating circles are fundamental, because they can contain subsets of vertices that fail the internal-adjacency criterion. An atomic circle is *not* considered to be self-dominating. A proper subset of an extreme star may be self-dominating.

$S$ is a *pure host-feasible subset of a fundamental circle* if (i) $S$ is host feasible with its largest degree vertex as host, and (ii) no member of $S$ is a member of any fundamental circle that does not contain all of $S$, unless the host of that circle is also a member of $S$. This second condition allows that $S$ may properly contain a second host of lesser degree and some of its children.

## 8.2 Preliminary Facts

The following lemma is a direct consequence of Theorem 27 and the definition of self-dominating circles.

**Lemma 34** *1. Every fundamental circle is a self-dominating circle if its host has index greater than that of each member.*

*2. Every extreme star is a self-dominating circle if its host has index greater than that of each child.*

*3. If vertices are indexed by increasing degree, the host of an extreme star has index greater than that of any child. (In case of ties, either vertex can be host, so we let the one with greater index be host.)*

The discovery of a host-feasible circle implies that a related fundamental circle exists.

**Lemma 35** *Let $X$ be any host-feasible circle, with host $x$.*

1. *$x$ is the host of a fundamental circle.*
2. *$X$ contains a largest "pure" subset $X'$ that is a subset of the fundamental circle hosted by $x$.*
3. *$X'$ contains a subset $X''$ that is self-dominating with leader $x$, when $x$ has index greater than that of each other member of $X'$.*

**Proof.**

(1) Host feasibility implies a bottleneck cut between $x$ and every vertex outside. Hence, $x$ cannot be a leaf of anything outside, nor can it be a leaf of anything inside (since it is of greatest degree), so $x$ must be a fundamental circle host.

(2) Consider the subset $Y \subset X$ consisting of vertices that are not actually members of $x$'s fundamental circle. By Theorem 27 collectively their adjacency ratio to $X$ must be $\leq 1/2$, so if we throw them all out of $X$, leaving $X'$, the result must still be host feasible.

(3) Next, consider any vertex in $X'$, other than $x$, that is not dominated. Throwing it out cannot increase the cut. If we repeatedly throw out non-dominated single vertices (other than $x$) until no more exist, then we must eventually be left with a self-dominated circle having at least one non-leader vertex, because $[X'] < [x]$ and if we threw out all of the non-leader vertices this way, we would be left with $[x]$, which would require raising the value of the cut. ∎

## 8.3 Algorithms to Find Self-Dominating Circles

### 8.3.1 Test-Leader

Suppose we have assigned an index ordering to the vertices – any ordering – and we wish to test whether a particular vertex can be a leader of a self-dominating circle. The test-leader algorithm does this in $O(m)$ time.[3]

The following description is for clarity and simplicity. Some minor enhancements are known to be possible that will be described later.

Suppose that the vertices are indexed $1, 2, \ldots, n$, and we want to test vertex $1 < k < n$ to see if there exists an SDC of which it is the leader.

**Algorithm:**

1. Begin with a copy of the network in which each vertex and all of its edges are colored *black*. We also assume that the total degree of each vertex is known.
2. Color all of the vertices 1,2,...,k *red* (all the candidates for a SDC led by $k$ begin red; as we prove they cannot be members we re-color them black).

[3]Actually, it can be done in time proportional to the number of edges adjacent to vertices of index $\leq$ that of the candidate leader, and in even less time under lucky circumstances. However, in the worst case, this is $O(m)$.

3. Initialize $redCount = k - 1$, and establish an empty *queue*.

4. Iterate through vertices $1, 2, \ldots, k-1$. For each such vertex $i$ do:

   (a) Initialize the red-degree of $i$ to 0.

   (b) Iterate through its edges. For each edge, if the neighbor is red, color the edge red and increment the red-degree of $i$ by the weight of the edge.

   (c) If the red-degree of $i$ is $\leq 1/2$ of its total degree, enqueue $i$ on the *queue*.

5. While the *queue* is not empty, remove the first index, say it is $i$. Subtract 1 from *redCount*. Color vertex $i$ black. Iterate through the edges of $i$: for every one that is red, color it black and subtract its weight from the red degree of the neighbor. If this edge color change causes the neighbor to go from red dominated to black dominated (in case of tie, black wins) then enqueue it on the *queue* (except for vertex $k$ – we don't keep track of it's red degree).

6. When the *queue* is empty, if $redCount = 0$, then $k$ is not the leader of any SDC. If $redCount > 0$, then $k$ is the leader of an SDC consisting of all the red vertices.

**Discussion of** TEST-LEADER It is not necessary that the queue used to hold red vertices that are not red-dominated be FIFO. Black-dominated vertices can be cancelled in any order, and the result is the same.

It is also possible to start this algorithm with only a *subset* of the vertices $1, 2, \ldots, k-1$ colored red, and then it determines whether an SDC exists that is a subset of the initial red set.

It may be possible to make it a little quicker on average by starting the red set not as all vertices of index $\leq k$, but filling it in, on demand, as all vertices of index $\leq k$ that are also connected to $k$, which may be far fewer, in a sparse network.

### 8.3.2 FIND-FIRST-LEADER

Using TEST-LEADER, it is straightforward to find the first leader in any ordered sequence of vertices, by testing each vertex, from $2, \ldots, n-1$, to see if it leads an SDC. This is the FIND-FIRST-LEADER algorithm.

**Discussion** If no leader $< n$ is found the whole algorithm takes $O(nm)$ time. If we are lucky and find one near the front of the sequence, it could take much less time. If there are a number of disjoint SDCs, we could get lucky and find the first in $O(\log n)$ invocations of TEST-LEADER by noting that self-dominating circles can be disjoint, and splitting the problem in two parts, etc. However, we have not been able to find an algorithm that always finds the first leader in less than $O(nm)$ time.

One algorithm that would be very nice to discover is one that tests in $O(m)$ time whether an SDC exists with a leader $\leq k$.

### 8.3.3 Modifying FIND-FIRST-LEADER for special conditions

It is easy to impose some special conditions on termination of FIND-FIRST-LEADER. In particular, in a later section, we will be interested in finding the first self-dominating-set that (i) contains a specific vertex as a member, and (ii) has a leader that is also dominated by the other members. It should be clear that checking for these kind of conditions can be added easily, without modifying the complexity of the algorithm described. We refer to the algorithm with special conditions of this sort as SPECIAL-FIND-FIRST-LEADER.

### 8.3.4 The FIND-ALL-LEADERS Algorithm

The FIND-FIRST-LEADER algorithm terminates as soon as it finds the first leader. Since this is already $O(nm)$, with exactly the same worst case complexity we can compute the first leader of every vertex in the network. The FIND-ALL-LEADERS algorithm computes a *first-leader table*: for every vertex it gives the leader of the first SDC that dominates that vertex. The first leader table essentially gives a tree, where every leader is an internal node, with all of its followers as children.

The procedure is to initialize the first leader table with NULL as the first leader of each vertex, from 1 to $n$. Then we use TEST-LEADER to test each of vertices $2, \ldots, n-1$ as leaders. Suppose that $k$ turns out to be a leader: then we iterate over the red vertices (other than $k$), and for any one with a NULL entry in the table, we enter $k$ as its first leader. At the end, we replace any remaining NULL entries with $n$.

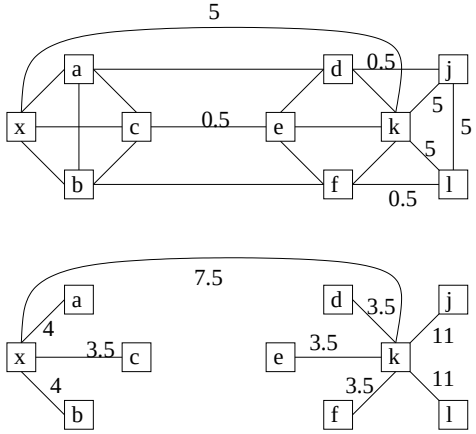
## 8.4 Filtering

Given the ability to find self-dominating circles, how might we use them to find fundamental circles? In this section we sketch a method for constructing the GH tree that first finds at least one candidate self-dominating circle containing each fundamental circle, and then filters each such set in order to discover any fundamental circle it contains. The method is incomplete because we cannot show a perfect filtering algorithm guaranteed to run in polynomial time. We describe an algorithmic approach that tends to organize the candidate vertices properly, identify some properties that would yield a correct algorithm if they could be computed efficiently, and consider the relationship of the filtering problem to a known NP-complete problem.

### 8.4.1 Method Outline

Begin by indexing all vertices in order of increasing degree. By Lemma 34 we know that every host of a fundamental circle is also the leader of an open SDC in this ordering. So, compute the first-leader table for the initial ordering, and then apply the filtering subroutine to every SDC, in order of increasing leader index. When the filtering routine finds a (possible) fundamental circle, it is contracted (which does not alter the first-leader table, except by simplifying it). The order of contractions is almost an optimal contraction sequence, in the sense of the formulation defined in Section 6.2. The only exception (assuming that filtering yields only correct answers) is that some internal vertices of low degree may have been

Figure 13: Example network and its GH tree




inverted, *i.e.* absorbed as children of hosts that should be their children. So, at the end of filtering each SDC, the GH tree is reconstructed from the contraction sequence, making any necessary corrections via the child-exclusion rule, as described in Section 6.4.1.

### 8.4.2 Group Packing

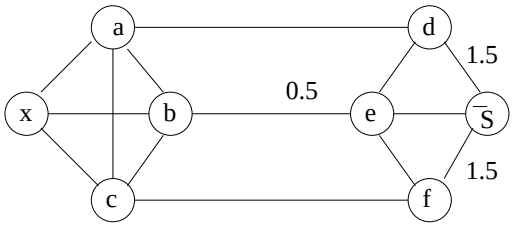
Our filtering approach utilizes a concept of a minimal-maximal self-dominating circle, closed with respect to hypothesized members of a fundamental circle. The circle is maximal in including all vertices dominated by its members, but minimal in not containing any smaller self-dominating circle of the same kind. An example will help to clarify. Figure 13 shows a network of 10 vertices, and its GH tree which has two internal nodes. (Unmarked edges have unit weight.) One possible vertex ordering consistent with increasing degree is shown below, with its first-leader table.

| name | $d$ | $e$ | $f$ | $b$ | $a$ | $c$ | $x$ | $j$ | $l$ | $k$ |
|---|---|---|---|---|---|---|---|---|---|---|
| index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| first leader | $c$ | $c$ | $c$ | $c$ | $c$ | $c$ | $x$ | $k$ | $k$ | $k$ |

Consequently, the first self-dominating circle passed to the filtering routine has leader $c$ and members $d, e, f, b, a$. This SDC does not actually contain a fundamental circle, which can be detected by the filterning routine. The second SDC to be passed to the filtering routine is a more interesting example with which to illustrate filtering. It has leader $x$ and members $d, e, f, b, a, c$.

Filtering begins by creating a reduced network, in order to simplify the problem. Let $S$ be the self-dominating circle to be filtered, with leader $x$. The filtering hypothesis is that $x$ is a host of a fundamental circle $X$, where $X \subseteq S$. The reduced network $G_S$ contracts all of $\overline{S}$ into a single vertex, and assigns weight 0 to edge $(x, \overline{S})$. The initial index order of $G_S$ is identical to that of the original network, except that $x$ is given index 1 ($\overline{S}$ remains at the end of the order). Figure 14 shows the reduced network, and its initial index ordering is

Figure 14: Reduced network




given below, together with the *closed* SDC first-leader table based on hypothetical host $x$.

| name | $x$ | $d$ | $e$ | $f$ | $b$ | $a$ | $c$ | $\overline{S}$ |
|---|---|---|---|---|---|---|---|---|
| index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| first leader | | $c$ | $c$ | $c$ | $c$ | $c$ | $c$ | |

Recall that a closed SDC is one in which we begin with one vertex that is always a member and exempt from the domination requirement, and then add other members each of which must be dominated by the complete membership. A first closed SDC can be found by a simple modification of the FIND-FIRST-LEADER algorithm. The table above does not show a first-leader for $x$ or $\overline{S}$ because they are the assumed hosts: the task of filtering is to determine which of the rest of the vertices in the reduced network belong on $x$'s side of the minimum cut between $x$ and $\overline{S}$.

The first group (from the left) is found by a process of iterative compaction and rotation. In the *compaction* step we take every vertex in the first SDC, plus every other vertex iteratively dominated by this set, and move them ahead in the ordering of all other vertices, otherwise retaining all relative orders. In the example, the first SDC with leader $c$ is already compacted to the left, and there are no other vertices in $G_S$ to be dominated by it. The candidate group is now $d, e, f, b, a, c$. In the *rotation* step we reorder the candidate group, moving its first member to the end, then recalculate and recompact the first SDC. The first-leader table of the rotated ordering is as follows.

| name | $x$ | $e$ | $f$ | $b$ | $a$ | $c$ | $d$ | $\overline{S}$ |
|---|---|---|---|---|---|---|---|---|
| index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| first leader | | $d$ | $d$ | $c$ | $c$ | $c$ | $d$ | |

After compaction we have:

| name | $x$ | $b$ | $a$ | $c$ | $e$ | $f$ | $d$ | $\overline{S}$ |
|---|---|---|---|---|---|---|---|---|
| index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| first leader | | $c$ | $c$ | $c$ | $d$ | $d$ | $d$ | |

Now the candidate group is just $b, a, c$ which does not dominate any of the other vertices in $G_S$. Two more rotations testing permutations $a, c, b$ and $c, b, a$ find no further reduction, so the first group is finalized as $b, a, c$. (In this example, $x \cup \{b, a, c\}$ is host feasible and a fundamental circle.)

The next group is found by repeating the rotate and compact operation on the next closed SDC (using $x$ *union* all previous groups as the base), etc., until the vertex ordering of $G_S$ has been completely partitioned into packed groups. Rotation and compaction are performed up to $k$ times on a candidate group of size $k$. The rule is that every vertex must be last in the group for one trial, in order to test whether it is necessary.

The group packing operation can be run from the left, based on $x$, as in the foregoing example, or it can be run from the right, based on $\overline{S}$. To do that, conceptually all we must do is reverse the initial ordering of vertices in $G_S$, perform the packing, and then reverse the ordering again. (In practice, the algorithm can just as easily run on the existing order, in reverse.)

A packed group ordering in the reduced network satisfies some interesting properties.

**Lemma 36** *Any packed group ordering computed from the left is unchanged if recomputed from the right, and vice versa.*

**Proof.** Consider an arbitrary group $W$ in a packed group ordering. Let $x'$ denote the union of $x$ and all groups to the left of $W$, and let $\overline{S}'$ denote the union of $\overline{S}$ and all groups to the right of $W$. Assuming that the group packing was performed from the left, then we know that no member of $W$ is dominated by $x'$, otherwise it would have been incorporated into an earlier group. We also know that every member of $W$ is dominated by $x' \cup W$, by the definition of a self-dominating circle, so no member of $W$ is dominated by $\overline{S}'$. Similarly, no member of $\overline{S}'$ is dominated by $x' \cup W$, and each must be dominated by $\overline{S}'$. Therefore, $W$ is stable under repacking from the right. ∎

Although the results of group packing from the left and right may differ for an arbitrary initial ordering of vertices, a group packing is stable under recomputation.

**Lemma 37** *If there exists a host-feasible $X \subset S$, then there are at least 2 groups in any group packed order.*

**Proof.** This lemma is a consequence of the repetition of rotation and compression in group packing. Since $x$ is the vertex of largest degree in $S$, and the outer algorithm contracts any smaller fundamental circles, $x$ will be the host of any such host-feasible circle, and we know that it must contain some self-dominating circle $X' \subset X$ based on $x$. Let $Y = S - X'$. If the initial candidate group is smaller than $S - x$, then the theorem holds. If the initial candidate group contains all vertices (other than $x$ and $\overline{S}$), then it also contains $Y$. In some rotation a member of $Y$ will be last in the group, and then a smaller group will necessarily be found and compacted left, resulting in at least two groups. ∎

**Corollary 38** *If there is only one group in a group packed order, then either $S$ is a fundamental circle hosted by $x$, or $S$ does not contain any host-feasible circle.*

Actually, there is a slightly cheaper but equivalent test for the condition of $S$ not properly containing a fundamental circle. If we compute the first leader table from the left, and find that the last vertex before $\overline{S}$ (call it $z$) is the first leader, then rotate $z$ to just after $x$ and

compute the first leader table from the right, and again discover that $z$ is the first leader, then the condition holds.

Define a *linear cut* to be any prefix of a complete vertex ordering. For example, $\{x, v_2, v_3, \ldots, v_i\}$ is a cut, and $\{v_{i+1}, v_{i+2}, \ldots, \overline{S}\}$ is its complement. A group packing *respects* a linear cut if it does not change the membership of its defining sets.

**Theorem 39** *Group packing respects any linear cut that is a fundamental circle.*

**Proof.** Suppose that $\{x, v_2, \ldots, v_i\}$ is a fundamental circle. Then, it is a self-dominating set, and does not dominate any other vertices in $G_S$. During compaction, no vertices to the right of $v_i$ will be members of any candidate group until all of $v_2, \ldots, v_i$ have been packed. ∎

### 8.4.3 A Recursive Approach

It should be clear that group packing tends to bring more adjacent vertices closer together in the ordering, and Theorem 39 implies that if we ever do get an ordering that respects a fundamental $x$-$\overline{S}$ cut, group packing will not disturb it. The situation suggests a recursive approach: Once we have a group packed ordering for the reduced network $G_S$, it is possible to define a "meta-problem" by contracting each group into a single vertex, retaining their relative index order, yielding a further reduced network $G_S^1$. Then we can apply the same or a different algorithm to $G_S^1$ that may establish a new order over its vertices. The result of solving the meta-problem can be transfered onto the vertex ordering of $G_S$ by re-indexing so that every group is still consecutive, but groups have the same relative order as the final ordering of the corresponding vertices of $G_S^1$.

To complete a correct algorithm of this kind, one of two things would be sufficient: (i) a way of "purifying" groups in $G_S$, or (ii) a way of finding the min $x$-$\overline{S}$ cut in $G_S$, given the corresponding min cut in $G_S^1$.

Suppose that we call a vertex "good" if it belongs to a fundamental circle hosted by $x$, and "bad" otherwise. Then, a *pure* group is one consisting only of good, or only of bad vertices. Observe that if all groups in a group packed ordering of $G_S$ are pure, then the groups can be arranged, without splitting any of them, to respect the fundamental circle hosted by $x$. Hence, if the vertices of the meta-problem are ordered such that its min cut is a linear cut, then that order transferred onto the groups of $G_S$ reveals the min cut as a linear cut. (And, the least linear cut can be found quickly.) So, if we can purify the groups in $G_S$, then we can do the same in $G_S^1$ and in its meta-problem $G_S^2$, etc., until there are only a small number of vertices in some highly reduced problem, whose min cut is obvious.

In the second case, we similarly get a recursive algorithm, though without the obligation to purify the groups at one level, before calling the next.

Unfortunately, we do not know of a polynomial time way of doing either (i) or (ii) that relies only on domination properties and does not essentially involve solving the min cut problem by a conventional path packing technique. Perhaps some value can be gained by considering possible recursive algorithms in more detail, and why they do not appear promising.

An easy theorem we will not bother to prove is that any mixed group (*i.e.* not pure) must contain at least two good vertices and two bad vertices, and any pure group must contain at least two vertices. Hence the size of a meta-problem is no more than 1/2 the size of the original problem, and the depth of recursion is no more than logarithmic in the size of $S$.

Without a certificate of optimality to prove that all groups are pure, or that the current ordering reveals the min cut as a linear cut, it appears that any algorithm must terminate by proving that no improvement is possible, perhaps by iterating to stability. For example, we might do group-packing on $G_S$, then recur on the meta-problem. When the recurrent application terminates, we transfer the ordering onto $G_S$, and group pack again. If the second packing does not change the ordering, we are done. Otherwise, contract the groups and recur on the meta-problem again. [4] Another possibility, to strengthen the algorithm, is to add some local exploratory permutation of groups, in order to split mixed groups into pure components. But the problem of showing an ordering of vertices which cannot be permuted in order to split any groups, or improve the least linear cut, is suspiciously similar to the NP-complete problem of Optimal Linear Arrangement [12].

An optimal linear arrangement of the vertices of a network is a linear ordering in which the total sum of linear cuts is minimal. We can define an $s$-$t$ OLA as the ordering with $s$ first and $t$ last that minimizes this sum. Clearly OLA reduces to $s$-$t$ OLA in $O(n^2)$ time, so $s$-$t$ OLA is also NP-complete. Adolphson and Hu [1] showed that an $s$-$t$ OLA respects the $s$-$t$ min cut. Unsurprisingly, if we could solve that problem we could filter correctly. Unfortunately, we cannot find any way of strengthening SDC group packing, except for just solving $s$-$t$ min cut by path packing, that guarantees purifying the ordering prefix to reveal the minimum cut between $x$ and $\overline{S}$ without necessarily establishing the $s$-$t$ optimal linear arrangement. There is a relationship between recursive group packing and optimal linear arrangement that is partially revealed by the following theorem. It may be possible to add additional criteria to the group packing algorithm in order to correctly purify groups, but it appears that such an algorithm must be worst-case as expensive as an algorithm for OLA.

**Theorem 40** *An $s$-$t$ optimal linear arrangement of the vertices of $G_S$, with $x$ and $\overline{S}$ as the two extreme vertices, is stable under recursive group packing.*

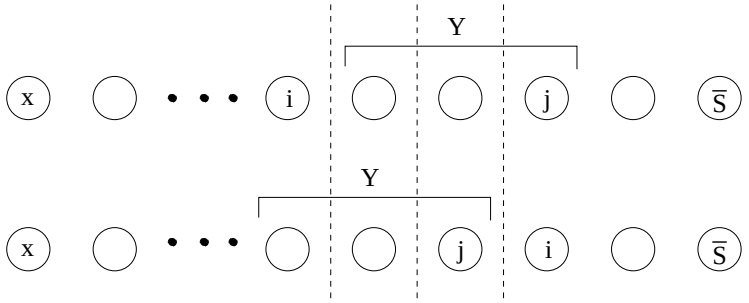
**Proof.** We will show that if at any "level" of recursion, if the ordering is not group packed, then it cannot be an optimal linear arrangement because an alternative arrangement has a smaller sum of linear cuts.

Consider an arbitrary level of recursion where the ordered vertices are $v_2, v_3, \ldots, v_k$, and the first and last vertices are $x$ and $\overline{S}$, as usual.

Assume without loss of generality that all levels below (*i.e.* the product of fewer contractions of packed groups) are group packed, but that the ordering at this level is not group packed. Then there exists another permutation $\pi$ that is group packed. Let $\pi(v_i)$ denote the index order of $v_i$ in this permutation, and assume without loss of generality that $\pi$ is conservative with respect to the original order, that is, the relative order of vertices differs only as much as is necessary to establish group packing. Let $v_j$ be the *rightmost* vertex in

[4] This particular recursive procedure often reveals the min cut between $x$ and $\overline{S}$ as a linear cut, but does not always do so.

Figure 15: Schema for Theorem 40




the original ordering that moves to the *left* of some other vertex in $\pi$, *i.e.* there exists $v_i$ such that $i < j$ but $\pi(v_i) > \pi(v_j)$. Let $v_i$ be the *rightmost* vertex in the original ordering that changes relative position with $v_j$ in ordering $\pi$. Let $Y = \{v_{i+1}, \ldots, v_j\}$ be the consecutive vertices following $v_i$ up through $v_j$ in the original ordering. The situation is illustrated in the upper part of Figure 15. By the definition of group packing (and the assumption that $\pi$ is conservative), strictly less than half of the degree of $v_i$ is to vertices of index $\leq \pi(v_j)$ in ordering $\pi$. Similarly, the set $Y$ has at least as much connection to the left as to the right in ordering $\pi$. Therefore, if we modify the original ordering by shifting $v_i$ to the right of $v_j$ then the linear cut preceding every vertex in $Y$ and $v_i$ strictly improves, and no other linear cut changes. The modified ordering is shown in the lower part of Figure 15, where the cuts that improve are shown by dashed vertical lines. If the level we have been considering is not the bottom-most (*i.e.* its vertices are contractions of vertices in the original network), then by the definition of group packing, all changed linear cuts in the bottom-most ordering are either unchanged or strictly improved. Therefore, the original vertex ordering at the bottom-most level cannot be an optimal linear ordering. ∎

# 9 A Randomized Algorithm to Find Bottleneck Cuts

Recently, a randomized or probabilistic approach has proven useful in developing new theoretical bounds and fast new algorithms for a range of problems of interest in computer science (see, *e.g.* Alon, Erdös and Spencer [3] and Motwani and Raghavan [25]). The first combinatorial algorithm for finding minimum cuts in a network that did not essentially rely on some version of Mengers theorem by constructing a saturating packing of paths or flows (or spanning forests, like Nagamochi and Ibaraki's maximum adjacency indexing and contraction algorithm [27]) was Karger's randomized contraction algorithm [21]. Karger's algorithm finds the global minimum cut (or all near-minimum cuts) in an undirected network with high probability by repeatedly selecting an edge at random, uniformly by weight, and contracting it. After $n - 2$ edges have been selected and contracted, the remaining network has only two vertices, representing one cut in the original network. The key insight is that a "typical" edge does not cross the global minimum cut. Under random selection there is never a higher expectation of selecting an edge that crosses the minimum cut than there is

of selecting one which does not (as long as there are more than a very few vertices), and usually the expectation of selecting such an edge is quite small. Hence the last remaining cut is at least as likely to be the minimum cut as any other, and it is feasible to run enough trials that the expectation of at least one trial finding the minimum cut is sufficiently high.

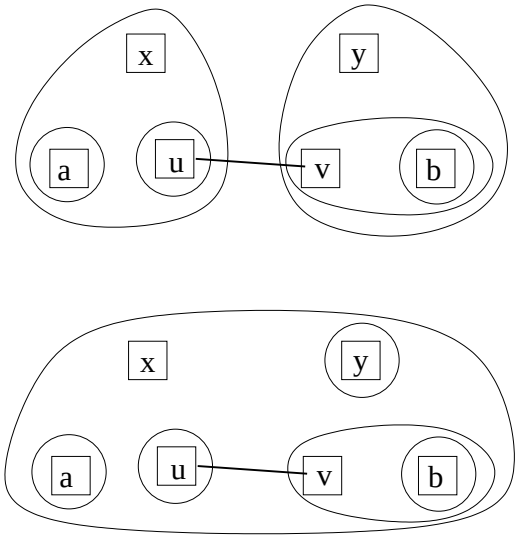
We are not aware of any non-path-saturating algorithms for finding any cuts other than global minimum and near-minimum cuts in a network—in particular, for finding $s$-$t$ minimum cuts or bottleneck cuts. Karger has devised several other randomized algorithms for finding cuts that include $s$-$t$ cuts [4, 19, 20], but they all employ random sampling to first reduce the number of edges and then apply a max-flow or tree packing algorithm to deterministically find cuts in the sampled network. Karger's randomized contraction algorithm cannot be applied within the Gomory-Hu meta-algorithm to find all $s$-$t$ min cuts because Gomory-Hu requires an $s$-$t$ min cut algorithm as a subroutine, not a global-min cut algorithm. The Gomory-Hu meta-algorithm leaves known leaf-cuts uncontracted, which a global-min cut algorithm might find again and again, in lieu of making progress by finding the unknown fundamental cuts that need to be resolved.

In this section we present a randomized approximation algorithm based on the optimal circle partition formulation which probabilistically discovers bottleneck cuts, even when all near-minimum cuts are leaf cuts, and thence the structure of the GH tree. It does not perform any path saturation or otherwise rely on a problem characterization following from Mengers theorem. Recall that a bottleneck cut is any minimum cut separating two vertices that is strictly less than the minimum of their degrees. Every non-trivial fundamental circle is a bottleneck cut separating its host from at least one vertex of larger degree on the outside.

## 9.1 Research Goals

The randomized algorithm described here is intended as an approximation algorithm. The goal is to find bottleneck cuts, which may not be detectable by Karger's algorithm. Analysis of the algorithm's expected performance on some special cases will show that the difficulty of finding a bottleneck cut increases dramatically as the ratio between the value of the cut and the degree of its host approaches 1. That is, let $X$ be a fundamental circle with host vertex $x$, and let $0 < \epsilon < 1$ be assigned such that $[X]/[x] = 1 - \epsilon$. Then, we will show that for some cases, the lower bound on the probability with which the algorithm can correctly find the fundamental circle $X$ drops exponentially as $\epsilon \to 0$. However, for a fixed $\epsilon$, the other factors involved in time complexity do not appear unfavorable, so we have sought to prove a lower bound that guarantees discovery of all bottleneck cuts within a sufficiently large $\epsilon$ in polynomial-time. So far, we have not been able to do so, but neither have we been able to construct a counter-example case for which the algorithm's performance is devastatingly bad. Our research efforts have focussed on proving a worst-case complexity bound for the algorithm, for *any input*, exploring possible worst-case inputs with highly non-random structure. It seems plausible that the algorithm may do a good job of finding most bottleneck cuts quickly, in some applications, although we do not have a good characterization of expected inputs in an application domain on which to evaluate this conjecture. In the following subsections we define the algorithm and present mathematical and empirical analysis that provide some basis for anticipating its behavior.

Figure 16: Edge shrinking




## 9.2 Description of Algorithm

The general method is to begin with a default circle partition of the input network, and repeatedly perform a sequence of randomly selected mergers in order to find an improvement to the current partition, or probabilistically prove that no such improvement exists. The final goal is a circle partition all of whose non-trivial fundamental cuts are within the approximation bound of optimal. This partition immediately yields an equivalently good approximation of the GH cut tree.

The algorithm begins with the *default circle partition* of the input network: this partition designates the vertex of highest degree as host of the circle containing all vertices, and every other vertex is host of its own circle, *i.e.* an atomic circle. The default partition corresponds to a tree with a single internal node. In essence, the algorithm begins with the null hypothesis that there are no bottleneck cuts, then seeks either to discover them, or to raise confidence in the correctness of the hypothesis to a sufficient level.

We distinguish between the actions of *contracting* a set of vertices and *shrinking* an edge. Contraction is the familiar operation of merging a set of vertices into one, eliminating self-loops and possibly merging multiple edges between remaining vertices. Shrinking is a similar operation, applied to a single pair of circles, that performs a circle merger but which does not lose information about the constituent vertices and their edges. Let $X, Y \subset V$ denote two disjoint circles with hosts $x, y$ respectively, joined by edge $(u, v)$ where $u \in X$, $v \in Y$. Assume without loss of generality that $[x] \geq [y]$. If we shrink $(u, v)$ the result is a circle $X' = X \cup Y$ where the host of $X'$ is $x$ and the circles immediately contained within $X'$ are all of those that were immediately contained within $X$ and $Y$ plus an atomic circle hosted by $y$. Figure 16 illustrates shrinking of such an edge between circles $X$ and $Y$.

Two key concepts are those of the *basic step* and the *active circle*. At the beginning of each basic step, the active circle is defined as the circle of least value among all *eligible*

*circles* (gradually defined below). It is assumed that there is an arbitrary total ordering over vertices, so that ties can be broken in favor of the circle hosted by the vertex of precedence. In a basic step, the active vertex merges with one of its neighbors. If an edge dominating at the active vertex exists, it is shrunk. Otherwise, one edge adjacent to the active circle is selected randomly, uniformly by weight, and shrunk. If the two circles merged were both eligible, the result is eligible; otherwise the resulting circle is not eligible. A *run* is a sequence of basic steps beginning with some initial circle partition, and ceasing when the result of shrinking an edge is host feasible, or when no eligible circles remain. At the start of each run, the mergers of the last run are erased, reseting the partition to its original condition, so each run is a randomized operation on the same starting partition. Eligibility of each initially existing circle to become active is determined at the beginning of a run.

The algorithm is organized into recursive invocations of the *check procedure*. A single invocation of the check procedure operates on a host feasible circle $X$, of host vertex $x$. This circle is a *candidate fundamental circle*, or simply "candidate," and $x$ is the candidate host. The hypothesis to be checked is that $X$ is a (subset of) a fundamental circle hosted by $x$, with no undiscovered substructure: that all the child circles within $X$ properly belong there, and no nested, host-feasible circle exists. To do so, we assign eligibility to all and only the child circles immediately contained within $X$, then temporarily violate partition legality by removing all children from $x$'s circle, so that it becomes atomic. (The atomic circle around $x$ is not eligible.) This is the starting partition for checking. The check procedure consists of a sequence of runs, beginning each time with the starting partition, until either

(i) a host-feasible subset $Y \subset X$ is found such that $Y$ does not iteratively dominate[5] $X - Y$, or

(ii) it can be concluded with sufficient probability that no such $Y$ exists, within the approximation bounds.

Case (i) has two subcases. In case (i-a) the host $y$ of $Y$ is not the same as the host $x$ of $X$. In this case $Y$ is a nested candidate so we suspend the check procedure for $X$ and begin a nested check procedure invocation that processes $Y$. After we are finished processing $Y$, which results in contraction of some vertices into a single vertex in the current partition, we resume processing $X'$, which differs from $X$ in that some of its members have been contracted. In case (i-b), $x$ is also the host of $Y$. In this case we reset $X'$ to be $Y$ union the iteratively dominated members of $X - Y$, and continue this invocation of the check procedure. In case (ii) we contract $X$ into a single vertex, adding this contraction operation to the end of a sequence $\mathcal{S}$ of contractions being accumulated, yielding a new circle partition, and return to processing any pending check procedure at a higher level.

Note: as will be seen later, all that is needed to control the goodness of approximation is to determine how many runs are necessary within the check procedure before case (ii) can be concluded. If a lower bound $0 < p < 1$ can be shown for the probability with which the check procedure discovers bottleneck cuts properly contained within the circle being checked, then the approximation bound can be guaranteed with high probability by performing no more

[5]A vertex set $X$ iteratively dominates a set $Y$ if $X$ dominates at least one vertex $v_1 \in Y$, $X + v_1$ dominates at least one vertex $v_2 \in Y - v_1$, etc.

than $O(\frac{-\log p}{p})$ runs in each invocation of the check procedure. But this is only practical if $1/p$ is polynomial in the size of the network, which we have not been able to show. In lieu of this, a practical implementation of this algorithm might use some adaptive feedback to try to estimate the likely utility of devoting more runs to a given invocation of the check procedure.

Given the foregoing definitions, the algorithm can be stated as follows.

1. Form the default circle partition of the input network in which every atomic circle is eligible. Call this $\mathcal{P}_0$.

2. Call the check procedure on $\mathcal{P}_0$ to discover any non-trivial fundamental circles contained within it, within the approximation bounds. The result is a single vertex formed by a sequence $\mathcal{S}$ of vertex set contractions.

3. Generate the (approximate) GH tree from the contraction sequence $\mathcal{S}$ performing local improvements with the correction algorithm described in Section 6.4.1.

## 9.3 Discussion

The basic insight motivating this algorithm is that while a "typical" edge may cross one or more bottleneck cuts, less than half of the edge weight of any minimum degree vertex will do so. Identifying leaf cuts is relatively easy; it is the bottleneck cuts between internal nodes of the GH tree that are harder to find, and easily damaged by improper shrinking. However, once we find the bottleneck cuts, all of the other cuts follow immediately. So, if we seek to find all $s$-$t$ cuts through random shrinking, we should restrict activity to the minimum degree vertex.

Heuristically, the anticipated action of the algorithm is that during a run it will tend to shrink GH tree leaf nodes into their host, or together with their siblings and then into the host, until a host feasible circle is found. During a run, some children properly belonging to a fundamental circle host may shrink an external edge to a member of another circle and thus "defect," while some vertices that are not children of a given fundamental circle may improperly "invade." More generally, during a run many vertices may become active, of which relatively few are members of the fundamental circle corresponding to the first host-feasible circle found. Starting a nested invocation of the check procedure on each host-feasible subset found reduces the scope of activity so as to most efficiently test the hypothesis that the host-feasible circle found contains only the members of a particular fundamental circle. Children that did not become active or defected will not become active during the check procedure, and invading vertices will have multiple chances to make a correct edge selection and then be eliminated as candidate members. Once we are confident in the membership of such a circle, it may be accepted as a new leaf vertex, through contraction, and search continues for the next depth-1 circle[6].

Let us allow that the algorithm as defined above is correct if it finds the correct GH tree in some possible execution, with correction during reconstruction from the contraction

[6]A depth-1 circle corresponds to a depth-1 subtree of the GH tree. All of the children of the host are atomic circles.

sequence. It should be easy to see that every vertex has at least one edge to another member of its smallest enclosing non-trivial fundamental circle, and so the active circle always has at least one correct edge to shrink. Hence, if every active circle shrinks a correct edge, eventually one correct subset of a depth-one fundamental circle will be found, and can safely be contracted. The only exception to this situation is if an internal node of the GH tree corresponds to a low degree vertex in the network which must become active before any depth-1 fundamental circle can be found. In this case we can argue that that internal vertex has at least one edge to shrink that does not invade a given depth-1 circle, or we can allow it to invade and rely on inversion correction during reconstruction to establish the correct GH tree.

Essentially, the argument is that there is a non-zero probability of finding a host-feasible subset of a depth-1 fundamental circle in every run where such a circle exists to be found (modulo errors that will be repaired by the correction algorithm). Consequently, there is a non-zero probability of correctly finding the GH tree.

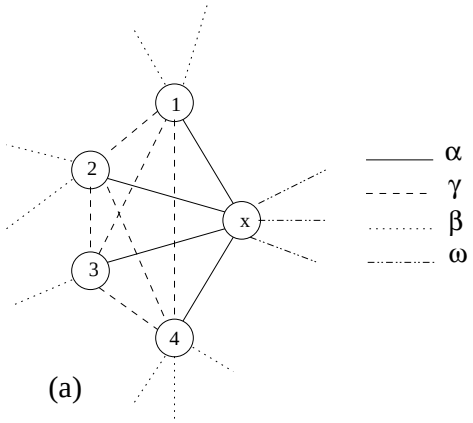
If we grant this weak form of correctness to the algorithm, the salient question becomes complexity: how many runs is it necessary to perform within the check procedure in order to identify case (ii), and conclude with high probability that its result cannot be improved? In subsequent sections we investigate lower bounds on the probability of correctly discovering a host-feasible (subset of a) fundamental circle during a run, and how that may be used to get a lower bound on expected performance of the entire algorithm.

## 9.4 Analysis

It is difficult to give a tight mathematical analysis of the behavior of this algorithm. The probability of a good or bad outcome in subsequent basic steps is too dependent on the outcome of prior steps, as well as on the particular structure of the input network. We have attempted to bound the worst-case behavior of the algorithm, for *any input network*, by a sequence of steps: (1) identifying a worst-case input for the check procedure among a restricted class, (2) proving a lower bound on the probability of that input being correctly processed by the check procedure, (3) generalizing that lower bound to a lower bound on the probability of a partially correct analysis of any input, (4) identifying from the probability bound an upper bound on the number of times the check procedure must be repeated to guarantee its results with high probability, and thus (5) upper bounding the size and number of check procedure invocations to correctly process an entire network, within a given approximation bound. For the restricted class in (1) we examined fundamental circles with no host-feasible proper subsets. We are able to show interesting lower bounds on two cases within the restricted class, but those bounds do not generalize to all fundamental circles with host-feasible proper subsets, so step (3) is incomplete. Some other bad cases have been identified for which the algorithm's empirically measured behavior is not too bad, but for which we do not have an analytical lower bound. Hence it is unknown whether the algorithm is worst-case polynomial time for all inputs (and sufficiently large $\epsilon$).

In the following we generally use $p_m$ and $p_s$ to indicate the probability of a "mistake" or "success" (*i.e.* no mistake) in a single basic step, and $P_m$ and $P_s$ the probability of one or more mistakes or no mistakes in a related sequence of steps. The exact definitions should be

Figure 17: Edge classification




clear from the context.

### 9.4.1 Classifying some hard cases

A run within the check procedure is said to make a *mistake* if it shrinks an edge between two circles when there exists a fundamental circle that contains all members of one, but not the other. In order to bound the complexity of the algorithm, we would like to be able to prove a lower bound on the probability that a run finds a depth-1 fundamental circle (or a host-feasible proper subset thereof) whenever one exists within the eligible scope of the current check procedure invocation, without making a relevant mistake. We will first study the case where the circle being checked actually is a depth-1 fundamental circle, with no more substructure.

It is convenient to classify the network edges into 4 kinds, as illustrated by Figure 17. With respect to a given depth-1 fundamental circle $X$, with host $x$, any edge between x and a leaf is an $\alpha$-edge, any edge between two leaves is a $\gamma$-edge, any edge from a leaf to a vertex outside of $X$ is a $\beta$-edge, and an edge between $x$ and any vertex outside $X$ is an $\omega$-edge. We let $\beta_i$ denote the sum of all beta edges adjacent to vertex $v_i$, and $\beta$ denote the sum of all beta edges; similarly for $\alpha$ and $\gamma$. In general, it is a mistake to shrink a $\beta$-edge, but safe to shrink an $\alpha$- or $\gamma$-edge. By working with the active vertex, we tend to avoid shrinking $\omega$-edges so for immediate purposes of analysis, we assume that the active circle never contains $x$.

**Lemma 41** *For any depth-1 fundamental circle, $\beta < \alpha$.*

**Proof.** Observe that $[x] = \alpha + \omega$ and $[X] = \beta + \omega$ so if $\beta \geq \alpha$ then $[X] \geq [x]$ and a necessary condition of optimality is not satisfied, so $X$ cannot be a fundamental circle. ∎

Recall that the unit of contraction (*i.e.* acceptance of substructure) for the algorithm is a host-feasible circle; the algorithm yields a correct answer if every circle it contracts is either a fundamental circle, or a subcircle of the smallest fundamental circle that contains two or more of its members. Hence, an extreme case for a single run is a fundamental circle

that does not contain any host-feasible subcircle: one where the entire circle must be shrunk without mistake before its host feasibility has been recognized. It turns out that restricting such a circle to have a highly homogeneous structure tends to raise the difficulty. Also, for a given $\epsilon$, the maximum difficulty is achieved when $\frac{[X]}{[x]} \to 1 - \epsilon$.

For probabilistic analysis we define a *normative case* fundamental circle to have such a homogeneous structure with a maximal $\beta/\alpha$ ratio. In the normative case, every leaf vertex has identical degree, and every edge in the same class has identical weight. Thus, if $k = |X| - 1$ is the number of leaves, then for all $i \leq k$, $\alpha_i = \alpha/k$, $\beta_i = \beta/k$, and $\gamma_i = 2\gamma/k$. Two distinct normative cases are of interest. In the normative *complete* case, every leaf has a $\gamma$-edge to every other. In the normative *cycle* case, every leaf has exactly two $\gamma$-edges, that connect the leaves in a cycle.

For convenience in analysis we normalize $\alpha = 1$. Note that the ratio $\frac{[X]}{[x]}$ is minimized as $\omega \to 0$, so for a given $\epsilon$ we can make a worst-case assumption that $\omega$ is arbitrarily small and dependently set the value of $\beta$ to $(1 - \epsilon)$. In general, the chance of making a mistake is increased as $\gamma$ decreases, so another pessimistic assumption is to pick a minimal feasible value for $\gamma$ for any $\alpha, \beta, k$, based on the following lemma.

**Lemma 42** *Within any depth-1 fundamental circle with no host-feasible proper subset,*

$$\gamma \geq \frac{(k-1)(\alpha - \beta)}{2}.$$

**Proof.** Let $X$ be a fundamental circle with host vertex $x$. One criterion for non-existence of a host-feasible proper subset is that for every leaf $v_i$,

$$\gamma_i \geq \max\left(\alpha_i - \beta_i,\ (\alpha - \beta) - (\alpha_i - \beta_i)\right) \tag{17}$$

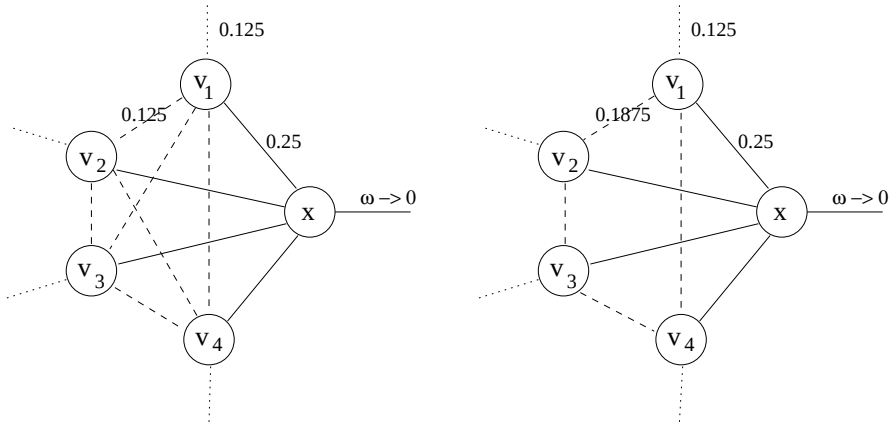
where $\gamma_i$ is the total weight of $\gamma$ edges adjacent to $v_i$, etc. If this criterion does not hold, then a host-feasible proper subset will exist. The first case, $\gamma_i \geq \alpha_i - \beta_i$, prevents a dominating edge at $v_i$, which would make $\{x, v_i\}$ host-feasible. The other case prevents $X - v_i$ from being host feasible. This criterion could be generalized further, to rule out other subsets of vertices, but this form is all we need to prove the lemma.
*Case 1:* For all leaves $v_i$, $(\alpha - \beta) - (\alpha_i - \beta_i) \geq \alpha_i - \beta_i$. Then,

$$\begin{aligned} \gamma &= 1/2 \sum_{i=1}^{k} \gamma_i \\ &\geq \frac{k(\alpha - \beta) - \sum(\alpha_i - \beta_i)}{2} \\ &\geq \frac{(k-1)(\alpha - \beta)}{2}. \end{aligned}$$

*Case 2:* For some leaf $v_j$, $(\alpha_j - \beta_j) > (\alpha - \beta)/2$. Observe that there can be only one such leaf, so every other $\gamma_i$ is lower bounded by $(\alpha - \beta) - (\alpha_i - \beta_i)$. Then,

$$\gamma = 1/2 \left( \gamma_j + \sum_{i \neq j} \gamma_i \right)$$

Figure 18: Normative complete and cycle cases




$$
\begin{aligned}
&\geq 1/2\left((\alpha_j-\beta_j)+\sum_{i\neq j}(\alpha-\beta)-(\alpha_i-\beta_i)\right)\\
&\geq 1/2\left((\alpha_j-\beta_j)+(k-1)(\alpha-\beta)-\sum_{i\neq j}(\alpha_i-\beta_i)\right)\\
&\geq 1/2\left((\alpha_j-\beta_j)+(k-1)(\alpha-\beta)+(\alpha_j-\beta_j)-(\alpha-\beta)\right)\\
&\geq \frac{(k-1)(\alpha-\beta)}{2}.
\end{aligned}
$$

∎

Consequently, in a normative case circle of $k$ leaves, with the minimal total weight of $\gamma$-edges,

$$\gamma_i = \frac{2\gamma}{k} = \frac{(k-1)(\alpha-\beta)}{k}.$$

If we pick the maximum $\beta/\alpha$ ratio compatible with a given $\epsilon$, we get

$$\gamma_i = \frac{(k-1)\epsilon}{k}.$$

In the normative complete case, each $\gamma$-edge has weight $\epsilon/k$. In a normative cycle circle of $k$ leaves, $\gamma_i$ is the same, but each individual $\gamma$-edge has weight $(k-1)\epsilon/2k$. Figure 18 illustrates the normative complete and normative cycle cases for $k = 4$ and $\epsilon = 1/2$.

### 9.4.2 Lower bounds on normative cases with no host-feasible proper subset

One interesting aspect of the normative cases is that they are amenable to analysis. Another is that we suspect that the normative cycle and normative complete cases are worst-case inputs to the check procedure within the class of fundamental circles with no host-feasible proper subset. However, since worse inputs outside of this class exist, we will not elaborate

the justification for this conjecture, but merely show some lower bounds on the probability of not making a mistake for these cases for the insight it yields into behavior of the algorithm.

As described in section 9.2, the basic step shrinks an edge that dominates at the active circle whenever one exists, and only selects an edge at random when a dominating edge does not exist. Precedence of dominating edges in this context is never a mistake, and hence can only contribute to raising the likelihood of a mistake-free run. In the following analysis, we assume that edges are always selected at random, with no precedence for dominating edges, which makes the analysis easier, and yields a lower bound somewhat more conservative than would actually apply. It is conjectured that the normative cycle case is the worst case when precedence is not given to dominating edges, while the normative complete case is worst for the actual algorithm that does select them with precedence.

**Lemma 43** *The probability of making $k$ steps with no mistakes in one run on a normative complete case fundamental circle of $k$ leaves and no host-feasible proper subset is*

$$P_{s1}(k) \quad \geq \quad \frac{e^{-(1+\epsilon)/\epsilon}}{2-\epsilon} \tag{18}$$

*hence*

$$P_{s1}(k) = \Omega(e^{-1/\epsilon}).$$

Before proving this lemma, we prove a simpler one.

**Lemma 44** *In a run of the check procedure on a normative complete case fundamental circle of $k$ leaves and no host-feasible proper subset, the probability of not making a mistake at step $i$, conditional on not having made a mistake in any of the previous $i-1$ steps, is lower bounded by*

$$\frac{1+(k-j)\epsilon}{2-\epsilon+(k-j)\epsilon} \quad \textit{where } j = 2^{\lg k - \lceil \lg(k-i+1) \rceil} \textit{ for } 1 \leq i \leq k.$$

**Proof.** In such a run, it is clear that the first $k/2$ active circles are all atomic, because the rule for picking an active circle is minimum degree, and the atomic leaves have smaller value than any merged circle containing two or more vertices. At step $\lceil k/2 \rceil + 1$ the active circle may still be atomic, *e.g.* if one of the prior steps selected an $\alpha$-edge, or it may be a circle formed by the merger of two leaves. More generally, let a "cluster" of size $i$ be the result of merging $i$ leaves in one circle. At step $1 \leq i \leq k$, the active circle will never be a cluster of more than

$$2^{\lg k - \lceil \lg(k-i+1) \rceil}$$

leaves, because if a cluster of larger than this size exists, then one of this size or smaller must also exist, and will have smaller value, and hence have priority as the active circle. For an active circle cluster of size $i$, the probability of not making a mistake is just the ratio of $\alpha$ and $\gamma$ weight, to total degree:

$$p_s(i) = \frac{i\alpha/k + i(k-i)2\gamma/(k(k-1))}{i\alpha/k + i\beta/k + i(k-i)2\gamma/(k(k-1))}.$$

The $\gamma$ weight of such a cluster is the weight of a single $\gamma$-edge ($2\gamma/(k(k-1))$) times $i(k-i)$. Normalize $\alpha = 1$, $\beta = 1-\epsilon$, and $\gamma = \epsilon(k-1)/2$. Then

$$\gamma_i = \epsilon(k-1)$$

and for $1 \leq i \leq k$ we can simplify to

$$p_s(i) = \frac{1+(k-i)\epsilon}{2-\epsilon+(k-i)\epsilon}.$$

Hence, the probability of not making a mistake at step $i$ is

$$\geq p_s(2^{\lg k - \lceil \lg(k-i+1) \rceil})$$

which gives the lemma. ∎

Observe that $i < j \Rightarrow p_s(i) \geq p_s(j)$ *i.e.* the probability of making a mistake monotonically increases with cluster size. Therefore, we can lower bound the probability of no mistakes in all $k$ steps of the benchmark run by the product of the probability of not making a mistake on the largest cluster that may be active at each step. This prepares us to prove the prior lemma.

**Proof of Lemma 43.** $P_{s1}(k)$ is lower bounded by the product of the probability of no-mistake at each step, conditional on no-mistake in any of the previous steps. From Lemma 44,

$$\begin{aligned} P_{s1} &\geq \prod_{i=1}^{k} p_s(k) \\ &\geq \prod_{i=1}^{k} p_s(2^j) \quad \text{where } j = \lg k - \lceil \lg(k-i+1) \rceil. \end{aligned}$$

Let $k = 2^r$, and $j$ be defined as above; observing that there are $2^{r-j-1}$ identical factors for each $0 \leq j < r$, and one additional factor for $i = k$, we can simplify.

$$\begin{aligned} P_{s1} &\geq p_s(k) \prod_{j=0}^{r-1} p_s(2^j)^{2^{r-j-1}} \\ &= p_s(k) \prod_{j=1}^{r} p_s(2^{j-1})^{2^{r-j}} \\ &= \frac{1}{2-\epsilon} \prod_{j=1}^{r} \left( \frac{1+(2^r-2^{j-1})\epsilon}{2-\epsilon+(2^r-2^{j-1})\epsilon} \right)^{2^{r-j}} \qquad (19) \\ &= \frac{1}{2-\epsilon} \prod_{j=1}^{r} \left( \frac{1+(2^r-2^{j-1})\epsilon}{2-\epsilon+(2^r-2^{j-1})\epsilon} \right)^{2^{r-j}} \\ &= \frac{1}{2-\epsilon} \prod_{j=1}^{r} \left( 1 - \frac{1+\epsilon}{2-\epsilon+(2^r-2^{j-1})\epsilon} \right)^{2^{r-j}} \end{aligned}$$

$$\geq \frac{1}{2-\epsilon}\prod_{j=1}^{r}\left(1-\frac{1}{\frac{\epsilon 2^j}{1+\epsilon}(2^{r-j}-1/2)}\right)^{2^{r-j}}$$
$$\geq \frac{1}{2-\epsilon}\prod_{j=1}^{r}\left(1-\frac{1}{\frac{\epsilon 2^j}{1+\epsilon}2^{r-j}}\right)^{2^{r-j}}$$

For further simplification, let

$$t=\frac{-(1+\epsilon)}{\epsilon 2^j}$$

and then take the limit as $k\to\infty$.

$$\lim_{k\to\infty} P_{s1} \geq \lim_{r\to\infty}\frac{1}{2-\epsilon}\prod_{j=1}^{r}\left(1+\frac{t}{2^{r-j}}\right)^{2^{r-j}}$$

Using the identity

$$e^t\left(1-\frac{t^2}{n}\right)\leq\left(1+\frac{t}{n}\right)^n$$

we get

$$\lim_{k\to\infty} P_{s1} \geq \lim_{r\to\infty}\frac{1}{2-\epsilon}\prod_{j=1}^{r}e^t\left(1-\frac{t^2}{2^{r-j}}\right)$$
$$\geq \frac{e^{\left(\lim_{r\to\infty}\sum_{j=1}^{r}t+\ln\left(1-\frac{t^2}{2^{r-j}}\right)\right)}}{2-\epsilon}$$

Using the identity $-x\geq\ln(1-x)$ we see that

$$\frac{-t^2}{2^{r-j}} \leq \ln\left(1-\frac{t^2}{2^{r-j}}\right)$$
$$t-\frac{t^2}{2^{r-j}} \leq t+\ln\left(1-\frac{t^2}{2^{r-j}}\right)$$

so we can simplify further.

$$\lim_{k\to\infty} P_{s1} \geq \frac{e^{\left(\lim_{r\to\infty}\sum_{j=1}^{r}t-t^2/2^{r-j}\right)}}{2-\epsilon}$$
$$= \frac{e^{\left(\lim\sum t-\lim\sum t^2/2^{r-j}\right)}}{2-\epsilon}$$
$$= \frac{e^{\left(\lim\sum -(1+\epsilon)/(\epsilon 2^j)-\lim\sum(1+\epsilon)^2/(\epsilon^2 2^{r+j})\right)}}{2-\epsilon}$$
$$= \frac{e^{\left(\frac{-(1+\epsilon)}{\epsilon}-0\right)}}{2-\epsilon}$$
$$= \frac{e^{-(1+\epsilon)/\epsilon}}{2-\epsilon} \qquad \blacksquare$$

Lemma 43 shows that the probability of no mistakes in a check procedure run on a normative complete case fundamental circle with no host-feasible proper subset is lower bounded by a constant, for any fixed $\epsilon$, but that this constant shrinks exponentially as $\epsilon \to 0$. The constant lower bound is a consequence of the restriction to no host-feasible proper subsets, which establishes a strong lower bound on $\gamma_i$, and the nature of the normative complete case which has as many $\gamma$ edges as possible, so the total weight of un-shrunken $\gamma$ edges decreases very slowly with each step. The normative cycle case concentrates $\gamma$ edges as much as possible, so that the total weight of un-shrunken $\gamma$ edges decreases quickly. We show that the analogous lower bound is not constant for the normative cycle case, but still reasonable for large $\epsilon$.

**Lemma 45** *The probability of making $k$ steps with no mistakes in one run on a $\gamma$-minimal, normative cycle case fundamental circle of $k$ leaves and no host-feasible proper subset, where the vertex ordering is random, is*

$$P_{s2}(k) = \Omega\left(e^{\frac{-6\Gamma(1-\epsilon)}{\epsilon\pi^2}}\left(\frac{k\epsilon}{2+\epsilon}\right)^{\frac{-6(1-\epsilon)}{\epsilon\pi^2}}\right)$$

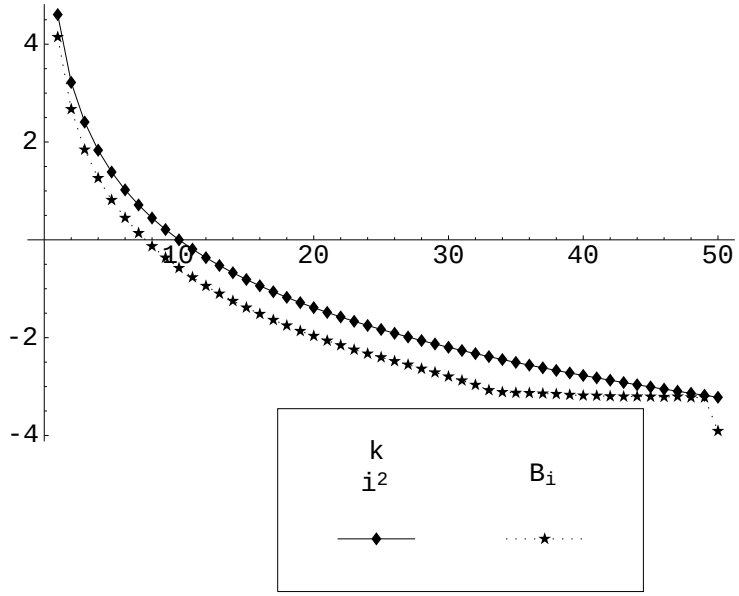
*where $\Gamma \approx 0.577$ is Euler's gamma constant.*

The proof requires first showing some bounds related to clustering behavior and the average probability of making a mistake at each step. In the proof of Lemma 43, we made the worst-case assumption that a run consists of $k/2$ size 1 clusters, $k/4$ size 2 clusters, $k/8$ size 4 clusters, etc. If we use this technique for Lemma 45 also, we will not be able to obtain the bound stated. In fact, a run of this type is extraordinarily unlikely, because formation of clusters is a random process that is roughly Poisson. Consider the following problem.

**Spotted ball problem:** There is an urn containing $k$ balls. Upon each ball is painted one spot. We follow a procedure with $k-1$ steps. At every step we first exhaustively search the urn and find a ball with no more spots than any other in the urn. This is the *selected* ball. We then shake the urn to mix it and draw another ball at random, uniformly from those in the urn, which is the *drawn* ball. We count the number of spots on the drawn ball, paint that many new spots on the selected ball, replace the selected ball in the urn, and discard the drawn ball. At the end of $k-1$ steps, the urn always contains 1 ball with $k$ spots. In fact, at the end of every step, the total number of spots on balls in the urn is always $k$. Let $B_i$ be a random variable whose value is the number of selected balls with $i$ spots. What is the expectation of $B_i$?

For large $k$ and a random vertex ordering, the number of clusters of size $i$ that become active should approximate $B_i$. The number of spots on a ball is analogous to the number of leaves in a cluster; selecting a ball is like picking the active circle; drawing another ball is like picking an edge of the active circle at random, conditional on only $\gamma$-edges being selected, and when vertex ordering is random and $\gamma$ edges are in a chain, so that expectation of selecting a gamma edge to any other existing cluster is approximately equal. We can show through analysis and experiment that $E[B_i] \leq k/i^2$. Suppose that $i < j \Rightarrow E[B_i] > E[B_j]$, *i.e.* the expectation is strictly decreasing. Observe that when the selected ball has $i$ spots,

Figure 19: Experimental $E[B_i]$



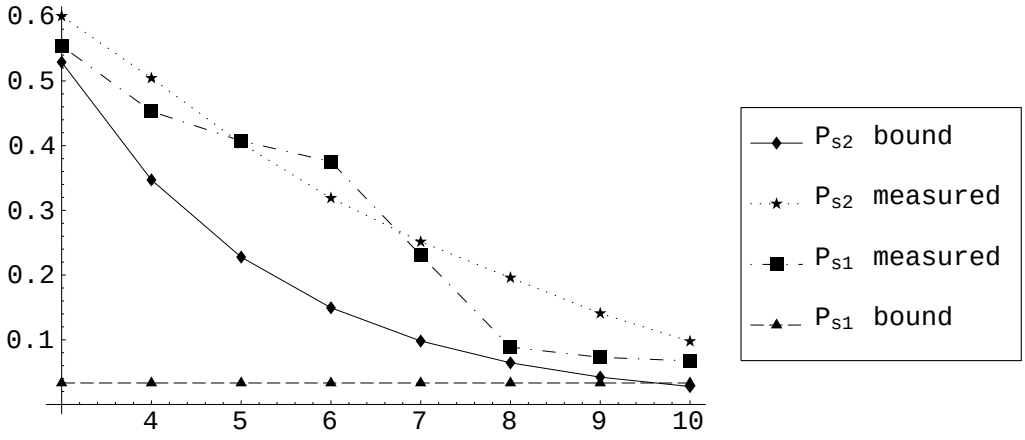

it will gain $\geq i$ additional spots. Let *round* $i$ consist of all the steps in which the selected ball has $i$ spots. Then, between the beginning of round $i$ and the end of round $2i-1$, only balls that had between $i$ and $2i-1$ spots at the beginning of this time period can be selected during it, and each will be selected at most once. Since there are a total of $k$ spots at any time, if they are all distributed among balls with between $i$ and $2i-1$ spots each, there can be no more than $k/i$ balls, divided among $i$ groups with the same number of spots. With the monotone-decreasing assumption, we get an upper bound of $k/i^2$ on the number of selected balls with $i$ spots, and experiment justifies this assumption. Figure 19 compares the average number of selected balls with $i$ spots to $k/i^2$ for $10^6$ trials and $k = 100$. The solid curve shows $\ln(k/i^2)$, and the dotted curve shows experimentally estimated $\ln(E[B_i])$. .

**Proof of Lemma 45.** During a check run on a normative cycle case fundamental circle, the number of active circles of size $1 \leq i \leq k/2$ is upper bounded by $k/i^2$. The total probability of success, $P_{s2}$, is the product of $p_s$ for each of the $k-1$ steps. As before, let $p_s(i)$ indicate the probability of success (not making a mistake) for a cluster of size $i$. Given the concavity of $1/i^2$ and the monotone decrease in $p_s(i)$, we can lower bound the product of $p_s$ for each of the first $k-1$ steps by: [7]

$$P_{s2}(k-1) \quad \geq \quad \left( \frac{\sum_{i=1}^{k/2} p_s(i)/i^2}{\sum_{i=1}^{k/2} 1/i^2} \right)^{k-1} \tag{20}$$

[7] We thank Prof. E. Bender for assistance with this asymptotic analysis.

Figure 20: Lower bounds and experimental values for normative cases




$$\geq \left( \frac{\sum \frac{i+(k-1)\epsilon}{i^2(i(2-\epsilon)+(k-1)\epsilon)}}{\sum 1/i^2} \right)^{k-1} \tag{21}$$

$$\geq \left( \frac{\sum 1/i^2 \left(1 - \frac{1-\epsilon}{i(2-\epsilon)+(k-1)\epsilon}\right)}{\sum 1/i^2} \right)^{k-1} \tag{22}$$

$$\geq \left( 1 - \frac{(1-\epsilon)\sum \frac{1}{i(2-\epsilon)+(k-1)\epsilon}}{\sum 1/i^2} \right)^{k-1} \tag{23}$$

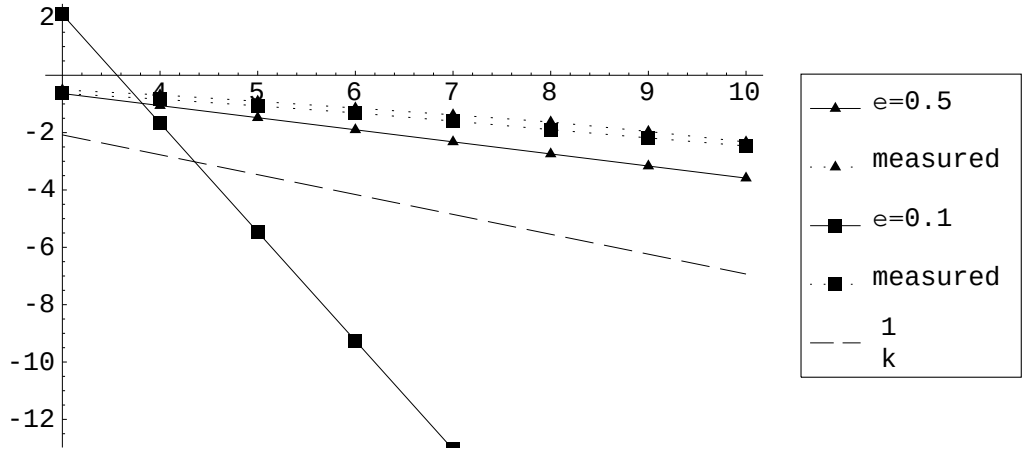
$$P_{s2}(k) = \Omega\left( \left( 1 - \frac{\frac{(1-\epsilon)}{k\epsilon}\sum_{i=1}^{k/2}\frac{1}{i} - \frac{2-\epsilon}{i(2-\epsilon+k\epsilon)}}{\sum 1/i^2} \right)^k \right) \tag{24}$$

$$= \Omega\left( \left( 1 - \frac{\frac{1-\epsilon}{k\epsilon}\left(\ln(k/2) - \ln(\frac{2+\epsilon}{2\epsilon}) + \Gamma\right)}{\pi^2/6} \right)^k \right) \tag{25}$$

$$= \Omega\left( e^{\frac{-6\Gamma(1-\epsilon)}{\pi^2\epsilon}} \left( \frac{k\epsilon}{2+\epsilon} \right)^{\frac{-6(1-\epsilon)}{\pi^2\epsilon}} \right) \quad \blacksquare \tag{26}$$

How tight are the lower bounds proven for the two normative cases considered? Figure 20 plots the lower bound given by Lemma 45 for the normative cycle case with a solid line, and that given by Lemma 43 for the normative complete case with a dashed line. Empirically measured values for these two cases are plotted by dotted (cycle) and dash-dot (complete) lines. For all curves $\epsilon = 0.5$ and both axes are in log scale, so the rightmost point shows $\ln P_s(2^{10})$. Every empirical point represents the average over $10^4$ trials. It can be seen that the theoretical lower bounds are not particularly tight, for small $k$. For the normative complete case, the constant lower bound is just a limit as $k \to \infty$. For the normative cycle case, the main reason for underestimating $P_{s2}$ is that our analysis did not take the selection

Figure 21: $\ln P_{s2}(k)$ versus $\lg k$: lower bound and experimental values




of dominating edges into account. If the dominating edge selection rule is turned off in the experimental measurement program, the results are much closer to the theoretical lower bound given by Lemma 45. Another reason for underestimating is that by using $k/i^2$ to bound cluster activation density in the proof of Lemma 45, we assumed that no $\alpha$-edges are contracted until the $k$'th step, which is too conservative. For large $k$, there are likely to be very few $\alpha$-edge contractions in the first $k/2$ moves, but towards the end of a run of $k$ steps their frequency increases, and hence the frequency of large active clusters (with their relatively high $p_m$) is actually somewhat lower than estimated in the analysis.

The bound given by Lemma 45 implies that, roughly, $P_{s2}(k) = \Omega(1/k)$ for $\epsilon > \frac{6}{\pi^2+6} \approx 0.378$, and $O(1/k)$ for lesser values of $\epsilon$. One might consider this approximation value to be a potential limit to practical effectiveness of the algorithm, since the lower bound decays exponentially as $\epsilon$ passes below this transition value. Figure 21 plots the natural log of the theoretical and empirical curves for the normative cycle case versus $\lg k$ for two different values of $\epsilon$. Again, each point is the average over $10^4$ trials. The curve for $\ln(1/k)$ versus $\lg k$ is also shown, for reference. It can be seen that the transition value of $\epsilon$ from practicality to impracticality is actually significantly lower in practice than predicted by Lemma 45.

## 9.5 Generalizing the Lower Bounds

Having shown lower bounds on the probability of a mistake-free run for normative case fundamental circles with no host-feasible proper subsets, we next consider to what extent these bounds may generalize to lower-bound the probability of discoving host-feasible circles in an arbitrary network.

### 9.5.1 Host activity and $\omega$-edges

In the proofs of Lemmas 43 and 45 we assumed that the active circle always contained only leaves of the target circle, never the host. This is true when the circle being checked really is a depth-1 fundamental circle, but is not necessarily true when substructure exists. That is, an active circle may contain the host of an as-yet undiscovered host-feasible circle. Even though all of the leaves of such a host should become active first, by the active circle selection rule, the host may become active if all of its children are in clusters with value larger than the current circle around the host. So we must consider whether this situation can result in behavior worse than that of the runs analyzed.

Consider a normative cycle case for $k = 8$, $\epsilon = 1/2$. This implies that the value of a cluster of $i$ leaves is

$$i/8 + i/16 + 7/16$$

which is $\leq 1$ for $i \leq 3$. So, even for very small $\omega$, in a mistake-free run the host will not become active before any 3-cluster. In the previous analyses, we assumed that that $\omega$ is very close to 0. In such a case, because its edge weight is almost entirely in the $\alpha$ category, if the host does become active, its contribution to any active circle's chance of making a mistake approaches 0. In fact, it tends to decrease the chance of a mistake; hence, not preventing the host from activation can only increase $P_s$.

One justification for specifying that $\omega \to 0$ in the previous analyses is that this allows us to set $\beta = (1-\epsilon)\alpha$, maximizing the $\beta/\alpha$ ratio at each leaf. Recall that $\epsilon$ is the approximation bound for the algorithm, *i.e.* we would like to guarantee finding any bottleneck cut such that $[X]/[x] \geq (1-\epsilon)$, with high probability. If we increase $\omega$, then, for a constant $\alpha = 1$ and $[X]/[x]$ ratio, $\beta$ must decrease; when $[X]/[x]$ is minimized, the chance of the host contributing to a mistake can only be raised by decreasing the chance of a leaf contributing to a mistake. The normative cases give us some understanding of the effects of the $\beta/\alpha$ ratio under no-host-feasible proper subset conditions where the the $\gamma$-minimal value given by Lemma 42 applies. The $\gamma$-minimal value is determined by the $\beta/\alpha$ ratio, not the $[X]/[x]$ ratio; for fixed $\beta/\alpha$, increasing $\omega$ thus decreases the $[X]/[x]$ ratio, and escapes the requirement for the algorithm to find the circle, if $\beta/\alpha \approx 1 - \epsilon$.

Hence, the extent to which $\omega$-edges can limit the generalizability of the normative cases results depends on whether increased $\omega$ weight can increase the chance of mistake more than it lowers the $[X]/[x]$ ratio, and hence the approximation tolerance requirement. Consider what happens to a normative cycle case of $\alpha = 1$, $\beta = 1 - \epsilon$, $\gamma = (k-1)\epsilon/2$, $\omega \to 0$, as we increase $\omega$; the implied approximation bound $\epsilon'$ which would require the algorithm to find the altered circle is

$$\begin{aligned}
(1 - \epsilon') &\geq \frac{\omega + \beta}{\omega + \alpha} \\
&\geq \frac{\omega + (1 - \epsilon)}{\omega + 1} \\
\epsilon' &\leq \frac{\epsilon}{\omega + 1}.
\end{aligned}$$

For $k = 8$ and $\epsilon = 1/2$, it is necessary to raise $\omega$ to 1 before the host can become active in a mistake-free run, but this lowers the implied approximation bound to $\epsilon' = 1/4$, while the

increased chance of mistake caused by additional $\omega$ is rather modest. It is not hard to see that the implied lower bounds given by Lemmas 43 and 45 for $\epsilon'$ decrease much faster than the actual probability of success, for fixed $\beta/\alpha$ and increased $\omega$.

Therefore, we can safely generalize bounds obtained for cases where $\omega \to 0$ to cases with arbitrary (non-negative) values of $\omega$.

### 9.5.2 Host-feasible proper subsets

The relatively strong lower bounds on the probability of a mistake-free run proven in Lemmas 43 and 45 apply only to special cases with no host-feasible proper subsets. This restriction implies strong bounds on the amount of $\gamma$-edge weight at each vertex, which is critical to the proofs. In the general case, a depth-1 fundmental circle may have many host-feasible proper subsets, so the amount of $\gamma$ weight at an arbitrary vertex need not be so great, allowing $p_s$ to be lower. On the other hand, when multiple host-feasible subsets exist the algorithm only need find and contract one in order to make progress. These two considerations simultaneously argue for a decrease and an increase in the probability with which one run of the check procedure discovers a circle that can correctly be contracted. It is not immediately obvious which is more important, and whether existence of host-feasible proper subsets is of advantage or disadvantage to the algorithm. Unfortunately, it also appears to be more difficult to give a mathematical analysis of the expected behavior of the algorithm for this kind of case than it was for the normative cases with no host-feasible proper subset.

Let us consider how a depth-1 fundamental circle $X$ with host-feasible proper subsets might be constructed in order to minimize $P_s$, which in this context is the probability of at least one host-feasible proper subset being shrunk into a single circle during one run of the check procedure applied to $X$. Suppose that $[X]/[x] = 1 - \epsilon$, and $Y \subset X$ is a host-feasible subset, then $[Y] > [X]$ and $[Y]/[x] < 1 - \epsilon$. The decreased value/host-degree ratio for the subset implies that that it will be less likely for the algorithm to shrink $Y$ without mistake than an equal size circle of larger ratio. In particular, any lower bound implied by an approximation bound of $\epsilon$ does not apply to this inner subcircle. Let $X' \subset X$ consist of all vertices that must be part of any host-feasible subset of $X$ ($x \in X'$, but it may otherwise be empty). If $|X'| > 1$, then $P_s$ may be lowered relative to a fundamental circle with no host-feasible proper subsets if active circles consisting of vertices not in $X'$ have a greater likelihood to defect, while the ratio $[X']/[x]$ is very close to 1, so that that circle itself is very unlikely to shrink without mistake.

For example, consider the chain of vertices in Figure 22. The edge $(x, v_1)$ has weight 1, and the weights decrease linearly to the right, down to $1 - \epsilon$. Properly, this is not a depth-1 fundamental circle because every edge from $x$ to $v_k$ is a dominating edge. However, if the check procedure were applied to the vertices shown, the probability of shrinking all and only these vertices together would be exponentially low, *if dominating edges were not preferentially selected.* At each step, conditional on no $\beta$-edge shrinkings so far, the rightmost circle would have smallest value so it would become active, and its probability of not shrinking a $\beta$-edge would be approximately 1/2, so the probability of no $\beta$-edge shrinking would be very roughly $\Omega(2^{-k})$. However, in order to get a single host-feasible circle with $x$ as host, it is only necessary that a prefix of $v_1, v_2, \ldots, v_k$ shrink into $x$, not the entire sequence, so it is

Figure 22: Chain configuration

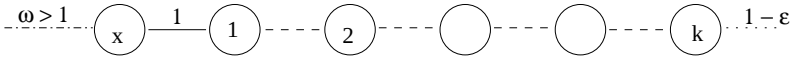
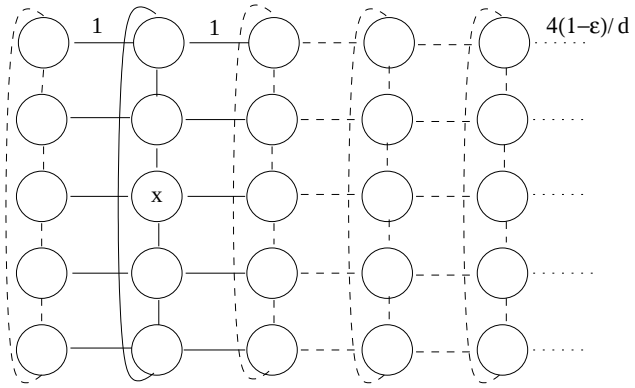



Figure 23: Cylinder configuration




not immediately obvious that this example can be generalized into a real bad case for the algorithm.

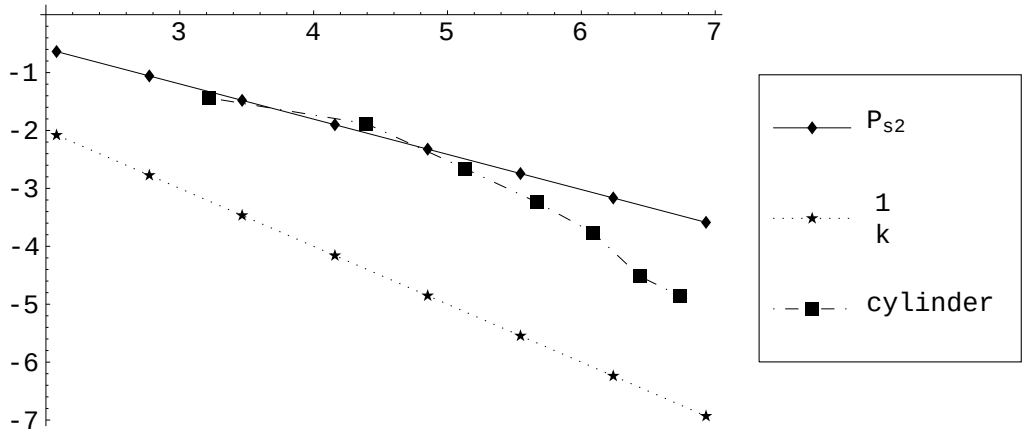
Figure 23 shows a similar configuration which is a single depth-1 fundamental circle, and which largely avoids the creation of dominating edges during a run of the check procedure. Instead of a single chain, vertices are arranged in a cylinder of length and circumference $d$ (equal to 5 in the figure), so $k = d^2$. In any circular cross-section of the cylinder, every vertex has 4 edges of approxmately equal capacity, but towards the right end of the cylinder the edge capacities gradually decrease so that the last cut (consisting of $\beta$-edges shown dotted) is of value $4(1 - \epsilon)$, while the degree of the host is $[x] = 4$ (there are $d$ vertices of degree 4 that may be host). Experiments with variations on this configuration, decreasing the weight of edges from left to right by various functions, and with various ratios of length to circumference, have discovered cases in which $P_s(k)$ decreases faster than either $P_{s1}$ or $P_{s2}$, for the same $\epsilon$, but not hopelessly fast. Figure 24 shows $\ln P_s(k)$ versus $\ln k$ for the worst cylinder configuration discovered and $\epsilon = 0.5$. The upper straight-looking line is the curve for $\ln P_{s2}(k)$ for the same $\epsilon$, and the lower line is the curve for $\ln 1/k$.

It is unknown whether even worse cases exist, or whether a strict lower bound better than $\ln 1/k$ can be proven for *any* depth-1 fundamental circle, and some $\epsilon < 1$. In subsequent sections we use $P_s(k)$ (or $\phi(k, \epsilon)$ when we wish to parameterize by $\epsilon$) to denote a hypothetical lower bound function for general depth-1 fundamental circles, whatever it is.

### 9.5.3 Multiple depth-1 fundamental circles

In our analysis up to this point, we have assumed that only vertices of one depth-1 fundamental circle could become active. In actual application, the active circle selection rule only

Figure 24: $P_s(k)$ for worst known cylinder configuration




guarantees that leaves first become active before their hosts; when the circle being checked has undiscovered substructure, the leaves of many different fundamental circles may become active before a mistake-free run can find the first host-feasible subset. Here we consider the effect of mistakes in one host-feasible circle on other circles. The next lemma says that we can use a lower bound on the probability of a mistake-free run to upper bound the probability of $i$ or fewer mistakes.

**Lemma 46** *In one run of the check procedure on a fundamental circle of of $k$ leaves, the probability of selecting a $\beta$-edge in $i$ or more separate steps is*

$$P_m(k, i) \leq (1 - P_s(k))^i.$$

**Proof.** The key insight is that selecting and shrinking a $\beta$-edge does not raise the probability of shrinking a $\beta$-edge in any subsequent step. It will tend to decrease because some $\beta$-capacity has been removed, raising the relative proportion of $\alpha$ and $\gamma$ weight. Therefore, the chance of $i$ or more $\beta$-edge selections is upper bounded by $i$ times the chance of at least one such selection. ∎

Next, consider application of the check procedure to the default circle partition, before any substructure has been identified. For simplicity, suppose that the network has only one bottleneck cut, so that the GH tree has exactly two depth-1 fundamental circles: one of $k$ leaves, and the other of $n - k - 2$ leaves. In the worst case, an adversary can break ties in selection of active circles so that $n - 3$ steps are necessary before either circle can be found, and any single mistake (*i.e.* $\beta$-edge shrinking) eliminates the bottleneck cut, making it impossible to find either circle. In this case, the chance of a no-mistake run guaranteed to find one circle is lower bounded by the product of a no-mistake run confined to either circle, which is minimized for $k = 2$ to $P_s(n/4 - 1)^2$.

Generalizing, consider a network whose GH tree has $h$ depth-1 fundamental circles, each of $k = n/h - 1$ leaves. Again, assume that an adversary breaks ties and that $\epsilon$ is small enough and other factors unfavorable enough that any $\beta$-edge shrinking erases the bottleneck cut between the two circles involved. In this case $hk-1$ steps are necessary to find the first circle, and the probability of a mistake-free run of this length is lower bounded by $P_s(n/h-1)^h$. However, with this many depth-1 circles, it is not necessary to have a mistake-free run in order that one circle be found. A single mistake, involving an active circle that is a subset of one depth-1 fundamental circle, can corrupt no more than two such circles. Let us say that an active circle $Y$ *invades* a depth-1 circle $X$ if $Y \cap X = \emptyset$ but the edge selected for merger joins a vertex in $Y$ to one in $X$. Similarly, a vertex $u \in X$ *defects* if it is part of an active circle $X' \subset X$ that merges with a circle containing a vertex not in $X$. Even if mistakes occur during a run, fundamental circle $X$ can still be found so long as none of its leaves defect, and it is not invaded.

**Lemma 47** *When the candidate circle contains $h$ depth-1 fundamental circles, a check procedure run successfully finds one of them with probability*

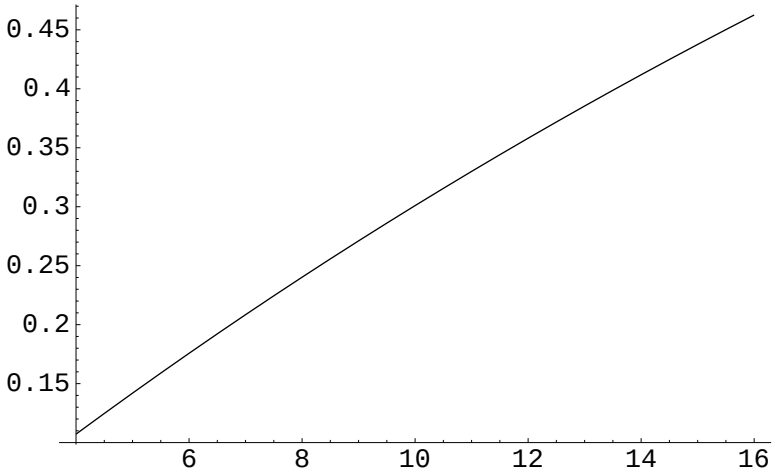
$$P_s(k,h) = \Omega\left(1 - h^{\frac{1}{2}\ln(1-\phi(k,\epsilon))}\right)$$

*where $P_s(k) = \Omega(\phi(k,\epsilon))$ is the lower bound on a mistake-free shrinking of any depth-1 fundamental circle of size $k$ satisfying approximation bound $\epsilon$.*

**Proof.** The situation is very similar to that described by the coupon collector's problem [25] for which it is known that if there is a large pool of $n$ different kinds of coupons, and at each step we draw a coupon equally likely to be of each kind, then the expected number of steps necessary to draw at least one coupon of each type is $n\ln n + O(n)$. When $\beta$-edges are evenly distributed, then each $\beta$-edge selection has an equal probability of resulting in the invasion of any of the other circles. (If they are not evenly distributed, then we can find one with relatively light $\beta$ weight where the situation is even more favorable, so without loss of generality assume that they are.) The differences that do exist, relative to the coupon collector's problem, all tend to decrease the probability of corrupting an uncorrupted circle, so we will ignore them, and assume that at each step, the chance of a $\beta$-edge selection involving a given circle is $\leq \frac{1-P_s(k)}{h}$, for each circle. In the worst case, $h/2$ $\beta$-edge shrinkings can corrupt all fundamental circles, so the probability of at least one circle being found without being affected by a mistake is $\geq 1/2$ if the total number of $\beta$-edge selections is $< \frac{h\ln h}{2}$. By Lemma 46, the probability of this many $\beta$-edge selections is $\leq (1-P_s(k))^{(\ln h)/2}$, consequently the probability of at least one circle not being involved in a defection or invasion is

$$\begin{aligned}
P_s(k,h) &\geq 1 - (1-P_s(k))^{\frac{1}{2}\ln h} \\
&\geq 1 - (1-\Omega(\phi(k,\epsilon)))^{\frac{1}{2}\ln h} \\
&\geq 1 - O((1-\phi(k,\epsilon))^{\frac{1}{2}\ln h}) \\
&= \Omega\left(1 - \left(e^{(1-\phi(k,\epsilon))}\right)^{\frac{1}{2}\ln h}\right) \\
&= \Omega\left(1 - h^{\frac{1}{2}\ln(1-\phi(k,\epsilon))}\right).
\end{aligned}$$

Figure 25: $P_{s2}(k,h)$ versus $h$ for $(k+1)h = 256$ and $\epsilon = 0.5$




Lemma 47 says that the lower bound on the probability of finding an undisovered fundamental circle by shrinking without any relevant mistakes actually goes up as more non-trivial fundamental circles meeting the approximation bound exist. Figure 25 plots $P_{s2}(k,h)$ for $4 \leq h \leq 16$ and constant $(k+1)h = 256$. By the argument given earlier, the minimum value ought to occur for $h = 2$, but the bound given by Lemma 47 is asymptotic, and not particularly accurate for small $h$.

Figure 26 shows the lower bound given by Lemma 47 for $2 \leq h \leq 6$, $(k+1)h \approx 64$ and $\epsilon = 1/2$, together with experimental measurement of the actual probability of success for the normative cycle cases with these values.[8] The solid line shows the theoretical lower bound, while the dotted line connects the experimental measurements.

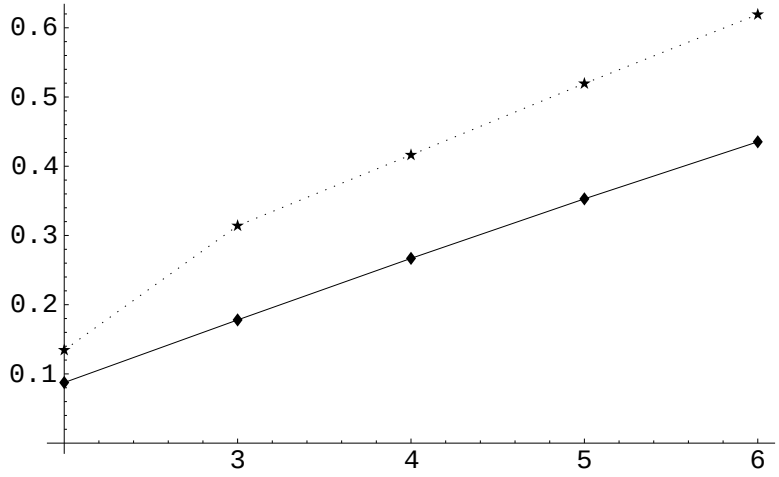
The key theoretical points are: (i) even when multiple depth-1 fundamental circles exist, the probability of success (here defined as fully merging one without relevant mistake) in a single run is no worse than $P_s(n/2)^2$, and (ii) $h = 2$ is the worst case, as the probability of success grows with $h > 2$.

## 9.6 Composition with Randomized Compression

Benczúr and Karger [4] give a nonuniform sampling algorithm for constructing a capacitated, undirected network $G'$ from a network $G$ such that, for any $0 < \epsilon < 1$, with high probability the value of any $s$-$t$ cut in $G'$ is $(1 \pm \epsilon)$ times its value in $G$. Moreover, the number of edges in $G'$ is $O(n \log n/\epsilon^2)$, and construction can be done in $O(m \log^3 n)$ time. It may be possible to reduce the running time of our randomized approximation algorithm by using their sampling algorithm to preprocess the input network. The idea is let any desired approximation bound $\epsilon = \epsilon_1 + \epsilon_2$, where $0 < \epsilon_1, \epsilon_2 < 1$. First compress $G$ to $G'$ within a tolerance of $\epsilon_1$ by the Benczúr-Karger algorithm, then approximate the $GH$ tree of $G'$ within a tolerance of $\epsilon_2$ by

[8] $10^5$ trials for each point.

Figure 26: Lower bound and empirical values of $P_{s2}(k,h)$ for $(k+1)h \approx 64$




our algorithm. The resulting GH tree approximates every $s$-$t$ cut within a factor of $\epsilon$, and the factor of $m$ in the complexity bound is replaced by a factor of $n \log n/\epsilon^2$. The practicality of this scheme is probably limited to fairly large $\epsilon$, primarily because our algorithm appears questionable for $\epsilon < 6/(\pi^2+6)$, but secondarily because the Benczúr and Karger algorithm does not significantly sparsify the network for $\epsilon \leq (m/\log n)^{-1/2}$.

## 9.7 Summary

In this section we presented a randomized algorithm for approximating the GH tree of an undirected network. The design of the algorithm is based on the idea of vertex degree domination, utilizing insights following from the new optimal circle partition formulation of the problem. We gave some characterization of the cases on which the algorithm is likely to perform well or badly. We analyzed its behavior on some extreme cases, showing that its probability of success drops exponentially, when faced with bottleneck cuts of host-degree/value ratio close to 1, indicating the algorithm is much more likely to yield approximate rather than exact solutions when such cuts exist. We showed that presence of host-feasible proper subsets can decrease the probability of success, but perhaps not too badly. We showed that presence of many fundamental circles tends to increase the probability of success, so that the worst-case input is a GH tree with a single internal link. It would be interesting to experiment with performance of variations of this algorithm on a real application, but that has not fallen within the scope of this research effort.

# 10 Conclusions

This report represents the highlights of a research effort that has substantially occupied its authors over several years. The guiding intuition has been that the overall structure of

minimum cuts in a network, as exemplified by the Gomory-Hu tree, has sufficient structure in terms of necessary inter-relationships that it should be possible to discover the entire GH tree with approximately the same worst-case computational complexity required to determine a single $s$-$t$ minimum cut. To date, the only known way to compute the GH tree remains the computation of $O(n)$ $s$-$t$ minimum cuts by way of maximum flows or similar technique. While specific goals of the effort evolved over time, much of our work concentrated on identifying and trying to exploit properties of minimum cuts that are independent of the path-saturation dualities identified by Mengers theorem and the max-flow min-cut theorem. We felt that the field of max-flow and similar algorithms is quite mature, and if a faster algorithm for the GH tree exists, it ought to rely on some method of finding the tree more directly, by bypassing the need to compute max flows, rather than computing individual $s$-$t$ flows faster. Otherwise unmentioned in this report is, for example, work we did on trying to save partial results in the context of a preflow-push max-flow algorithm, so as to compute GH subtrees in the same time as a single $s$-$t$ cut, which was not successful.

While the ultimate goal of a revolutionary new minimum cut algorithm was not achieved, this research effort did accomplish several notable positive results, which were described. In broad terms, we made a systematic study of vertex degree domination properties and their relation to minimum cuts. We gave a number of lemmas relating the properties of network vertices and edges to those of GH tree nodes and links. In close study of the maximum adjacency indexing algorithm, we proved that monotone-increasing indexings give strong lower bounds on connectivity in cases that had not previously been identified. We gave new, equivalent problem formulations for the GH tree, and proved the first necessary and sufficient optimality conditions for the problem that do not rely on any Menger-like theorem. We described several sensible heuristics and algorithmic approaches based on these new problem characterizations, including favoring edge contraction, computation of self-dominating sets, and randomized shrinking from the minimum-value circle. Although we strove for theoretical results and did not attempt to study performance in practice, it is possible that some of these approaches may be of practical benefit for some problems.

# References


[1] Adolphson, D., and Hu, T. C. Optimum linear ordering. *SIAM Journal on Applied Mathematics 25*, 3 (1973), 403–423.

[2] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. *Network Flows*. Prentice Hall, 1993.

[3] Alon, N., and Spencer, J. H. *The Probabilistic Method*. Wiley, 1992.

[4] Benczúr, A. A., and Karger, D. R. Approximating $s$-$t$ minimum cuts in $\tilde{O}(n^2)$ time. In *Proceedings of the 28th ACM Symposium on Theory of Computing* (May 1996), ACM Press, pp. 47–55.

[5] Chekuri, C. S., Goldberg, A. V., Karger, D. R., Levine, M. S., and Stein, C. Experimental study of minimum cut algorithms. In *Proceedings of the Eighth Annual*

*ACM-SIAM Symposium on Discrete Algorithms* (New Orleans, Louisiana, 5–7 Jan. 1997), pp. 324–333.

[6] Dantzig, G. B., and Fulkerson, D. R. Minimizing the number of tankers to meet a fixed schedule. *Naval Research Logistics Quarterly 1* (1954), 217–222.

[7] Dinits, E. A. Algorithm for solution of a problem of maximum flow in a network with power estimation. *Soview Mathematics Doklady 11* (1970), 1277–1280.

[8] Edmonds, J. Edge-disjoint branchings. In *Combinatorial Algorithms*, R. Rustin, Ed. Algorithmics Press, New York, 1972, pp. 91–96.

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

[10] Gabow, H. N. A matriod approach to finding edge connectivity and packing arborescences. *Journal of Computer and System Sciences 50*, 2 (April 1995), 259–273.

[11] Gabow, H. N., and Manu, K. S. Packing algorithms for arborescences (and spanning trees) in capacitated graphs. *Mathematical Programming 82* (1998), 83–109.

[12] Garey, M. R., and Johnson, D. S. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[13] Goldberg, A. V. Recent developments in maximum flow algorithms. Tech. Rep. 98-405, NEC Research Institute, April 1998.

[14] Goldberg, A. V., and Rao, S. Beyond the flow decomposition barrier. In *Proc. 38th IEEE annual sumposium on foundations of compter science* (1997), pp. 32–35.

[15] Goldberg, A. V., and Tarjan, R. E. A new approach to the maximum flow problem. *Journal of the Association of Computing Machinery 35* (1988), 921–940.

[16] Gomory, R. E., and Hu, T. C. Multi-terminal network flows. *Journal of SIAM 9*, 4 (1961), 551–570.

[17] Hu, T. C. Optimum communication spanning trees. *SIAM Journal on Computing 3*, 3 (1974), 188–195.

[18] Karger, D. R. Global min-cuts in $\mathcal{RNC}$, and other ramifications of a simple min-cut algorithm. In *Proceedings of the 4th ACM Symposium on Discrete Algorithms* (1993).

[19] Karger, D. R. *Random Sampling in Graph Optimization Problems*. PhD thesis, Stanford University, Stanford, CA, 1995.

[20] Karger, D. R. Minimum cuts in near-linear time. In *Proc. 29th Annual ACM Symposium on Theory of Computing* (1996), pp. 56–64.

[21] Karger, D. R., and Stein, C. An $o(n^2)$ algorithm for minimum cuts. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing* (New York, NY, May 1993), ACM, pp. 757–65.

[22] Karger, D. R., and Stein, C. A new approach to the minimum cut problems. *Journal of the ACM 43*, 4 (July 1996), 601–40.

[23] Kotzig, A. Súvislost' a pravideliná súvislost' konečných grafor. *Vysoká Škola Ekonomická* (1956).

[24] Menger, K. Zur allgemeinen Kurventheorie. *Fund. Math. 10* (1927), 96–115.

[25] Motwani, R., and Raghavan, P. *Randomized Algorithms*. Cambridge University Press, 1995.

[26] Nagamochi, H., and Ibaraki, T. Computing edge-connectivity in multiple and capacitated graphs. In *Algorithms. International Symposium SIGAL '90 Proceedings* (Berlin, Aug 1990), T. Asano, T. Ibaraki, H. Imai, and T. Nishizeki, Eds., Springer-Verlag, pp. 12–20.

[27] Nagamochi, H., and Ibaraki, T. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics 5*, 1 (February 1992), 54–66.

[28] Nagamochi, H., and Ibaraki, T. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica 7*, 5-6 (1992), 583–96.

[29] Padberg, M., and Rinaldi, G. An efficient algorithm for the minimum capacity cut problem. *Mathematical Programming 47* (1990), 19–36.

[30] Stoer, M., and Wagner, F. A simple min cut algorithm. In *Algorithms - ESA '94. Second Annual European Symposium Proceedings* (1994), V. Leeuwen, Ed., Springer-Verlag, pp. 141–7.