

# Packing multiway cuts in capacitated graphs

Siddharth Barman\*

Shuchi Chawla†

## Abstract

We consider the following “multiway cut packing” problem in undirected graphs: given a graph  $G = (V, E)$  and  $k$  commodities, each corresponding to a set of terminals located at different vertices in the graph, our goal is to produce a collection of cuts  $\{E_1, \dots, E_k\}$  such that  $E_i$  is a multiway cut for commodity  $i$  and the maximum load on any edge is minimized. The load on an edge is defined to be the number of cuts in the solution containing the edge. In the capacitated version of the problem the goal is to minimize the maximum *relative* load on any edge—the ratio of the edge’s load to its capacity. Multiway cut packing arises in the context of graph labeling problems where we are given a partial labeling of a set of items and a neighborhood structure over them, and the goal, informally stated, is to complete the labeling in the most consistent way. This problem was introduced by Rabani, Schulman, and Swamy (SODA’08), who developed an  $O(\log n / \log \log n)$  approximation for it in general graphs, as well as an improved  $O(\log^2 k)$  approximation in trees. Here  $n$  is the number of nodes in the graph. We present the first constant factor approximation for this problem in arbitrary undirected graphs. Our LP-rounding-based algorithm guarantees a maximum edge load of at most  $8\text{OPT} + 4$  in general graphs. Our approach is based on the observation that every instance of the problem admits a laminar solution (that is, no pair of cuts in the solution crosses) that is near-optimal.

## 1 Introduction

We study the *multiway cut packing* problem (MCP) introduced by Rabani, Schulman and Swamy [8]. We are given  $k$  instances of the multiway cut problem in a common graph, each instance being a set of terminals at different locations in the graph. Informally, our goal is to compute nearly-edge-disjoint multiway cuts for each of the instances. More precisely, we aim to minimize the maximum number of cuts that any single edge in the graph belongs to. In the weighted version of this problem edges have capacities; the goal is to minimize the maximum relative load of any edge, where the relative load of an edge is the ratio of the number of cuts it belongs to and its capacity.

Multiway cut packing belongs to the following class of graph labeling problems. We are given a partially labeled

set of  $n$  items with a weighted graph that encodes similarity information over them. An item’s label is a string of length  $k$  where each coordinate of the string is either drawn from an alphabet  $\Sigma$ , or is undetermined. Roughly speaking, the goal is to complete the partial labeling in the most consistent possible way. Note that completing a single coordinate of each item label is like finding what we call a “set multiway cut”—for  $\sigma \in \Sigma$  let  $S_\sigma^i$  denote the set of nodes for which the  $i$ th coordinate is labeled  $\sigma$  in the partial labeling, then a complete and consistent labeling for coordinate  $i$  is a partition of the graph into  $|\Sigma|$  parts such that the  $\sigma^{\text{th}}$  part contains the entire set  $S_\sigma^i$ . The cost of the labeling for a single pair of neighboring items in the graph is measured by the Hamming distance between the labels assigned to them. The overall cost of the labeling can then be formalized as a certain norm of the vector of (weighted) edge costs.

Different choices of norms for the overall cost give rise to different objectives. For example, minimizing the  $\ell_1$  norm or the sum of the edge costs reduces to finding  $k$  minimum set multiway cuts. Each set multiway cut instance can be reduced to a minimum multiway cut instance by simply merging all the items in the same set  $S_\sigma$  into a single node in the graph, and can therefore be approximated to within a factor of 1.5 [2]. On the other hand, minimizing the  $\ell_\infty$  norm of edge costs (equivalently, the maximum edge cost) becomes the set multiway cut packing problem. Formally, in this problem, we are given  $k$  set multiway cut instances  $S^1, \dots, S^k$ , where each  $S^i = S_1^i \times S_2^i \times \dots \times S_{|\Sigma|}^i$ . The goal is to find  $k$  cuts, with the  $i$ th cut separating every pair of terminals that belong to sets  $S_{j_1}^i$  and  $S_{j_2}^i$ ,  $j_1 \neq j_2$ , such that the maximum (weighted) cost of any edge is minimized. When  $|S_j^i| = 1$  for all  $i \in [k]$  and  $j \in \Sigma$ , this is the multiway cut packing problem.

To our knowledge Rabani et al. [8] were the first to consider the multiway cut packing problem and provide approximation algorithms for it. They used a linear programming relaxation of the problem along with randomized rounding to obtain an  $O(\frac{\log n}{\log \log n})$  approximation, where  $n$  is the number of nodes in the given graph<sup>1</sup>. This approximation ratio arises

\*sid@cs.wisc.edu, University of Wisconsin - Madison. Supported in part by NSF grant CCF-0643763.

†shuchi@cs.wisc.edu, University of Wisconsin - Madison. Supported in part by NSF awards CCF-0643763 and CCF-0830494.

<sup>1</sup>Rabani et al. [8] claim that the same approximation ratio holds for the set multiway cut packing problem as well. However their approach of merging nodes with the same attribute values (similar to what we described above for minimizing the  $\ell_1$  norm of edge costs) does not work in this case. Roughly speaking, if nodes  $u$  and  $v$  have the same  $i$ th attribute, and nodes  $v$  and  $w$  have the same  $j$ th attribute, then this approach merges all three nodes,

from an application of the Chernoff bounds to the randomized rounding process, and improves to an  $O(1)$  factor when the optimal load is  $\Omega(\log n)$ . When the underlying graph is a tree, Rabani et al. use a more careful deterministic rounding technique to obtain an improved  $O(\log^2 k)$  approximation. The latter approximation factor holds also for a more general multicut packing problem. One nice property of the latter approximation is that it is independent of the size of the graph, and remains small as the graph grows but  $k$  remains fixed. Then, a natural open problem related to their work is whether a similar approximation guarantee independent of  $n$  can be obtained even for general graphs.

**Our results & techniques.** We answer this question in the positive. We employ the same linear programming relaxation for this problem as Rabani et al., but develop a very different rounding algorithm. In order to produce a good integral solution our rounding algorithm requires a fractional collection of cuts that is not only feasible for the linear program but also satisfies an additional good property—the cut collection is laminar. In other words, when interpreted appropriately as subsets of nodes, no two cuts in the collection “cross” each other. Given such an input the rounding process only incurs a small additive loss in performance—the final (absolute) load on any edge is at most 3 more than the load on that edge of the fractional solution that we started out with. Of course the laminarity condition comes at a cost—not every fractional solution to the cut packing LP can be interpreted as a laminar collection of cuts (see, e.g., Figure 5). We show that for the multiway cut problem any fractional collection of cuts can be converted into a laminar one while losing only a multiplicative factor of 8 and an additive  $o(1)$  amount in edge loads. Therefore, for every edge  $e$  we obtain a final edge load of  $8\ell_e^{\text{OPT}} + 4$ , where  $\ell_e^{\text{OPT}}$  is the optimal load on the edge. We only load edges with optimal load at least 1, so this also implies a multiplicative 12 approximation.

In the full version of this paper [1] we show that our laminarity based approach proves even more powerful in the special case of the *common-sink s-t cut packing* problem. In this special case every multiway cut instance has only two terminals and all the instances share a common sink  $t$ . We use these properties to improve both the rounding and laminarity algorithms, and ensure a final load of at most  $\ell_e^{\text{OPT}} + 2$  for every edge  $e$ . This special case is also NP-hard and so our guarantee is nearly the best possible.

**Related work.** Problems falling under the general framework of graph labeling as described above have been studied in various guises. The most extensively studied special case, called label extension, involves partial labelings in which every item is either completely labeled or not labeled at all. When the objective is to minimize the  $\ell_1$

norm of edge costs, this becomes a special case of the metric labeling and 0-extension problems [6, 3, 4, 5]. (The main difference between 0-extension and the label extension problem as described above is that the cost of the labeling in the former arises from an arbitrary metric over the labels, while in the latter it arises from the Hamming metric.)

When the underlying graph is a tree and edge costs are given by the edit distance between the corresponding labels, this is known as the tree alignment problem. It has been studied widely in the computational biology literature and arises in the context of labeling phylogenies and evolutionary trees. This version is also NP-hard, and there are several PTASes known [12, 11, 10]. Ravi and Kececioğlu [9] introduced and studied the  $\ell_\infty$  version of this problem, the bottleneck tree alignment problem, providing an  $O(\log n)$  approximation for it. A further special case of the label extension problem under the  $\ell_\infty$  objective, where the underlying tree is a star with labeled leaves, is known as the closest string problem. This problem is also NP-hard but admits a PTAS [7].

As mentioned above, the multiway cut packing problem was introduced by Rabani, Schulman and Swamy [8]. Rabani et al. also studied the more general multicut packing problem (where the goal is to pack multicuts so as to minimize the maximum edge load) as well as the label extension problem with the  $\ell_\infty$  objective. They developed an  $O(\log^2 k)$  approximation for multicut packing in trees, and an  $O(\log M \frac{\log n}{\log \log n})$  in general graphs. Here  $M$  is the maximum number of terminals in any one multicut instance. For the label extension problem they presented a constant factor approximation in trees, which holds even when edge costs are given by a fairly general class of metrics over the label set (including Hamming distance as well as edit distance).

## 2 Definitions and results

Given a graph  $G = (V, E)$ , a *cut* in  $G$  is a subset of edges  $E'$ , the removal of which disconnects the graph into multiple connected components. A (2-way) *vertex partition* of  $G$  is a pair  $(C, V \setminus C)$  with  $\emptyset \subsetneq C \subsetneq V$ . For a set  $C$  with  $\emptyset \subsetneq C \subsetneq V$ , we use  $\delta(C)$  to denote the cut defined by  $C$ , that is,  $\delta(C) = \{(u, v) \in E : |C \cap \{u, v\}| = 1\}$ . We say that a cut  $E' \subseteq E$  separates vertices  $u$  and  $v$  if  $u$  and  $v$  lie in different connected components in  $(V, E \setminus E')$ . The vertex partition defined by set  $C$  separates  $u$  and  $v$  if the two vertices are separated by the cut  $\delta(C)$ . Given a collection of cuts  $\mathcal{E} = \{E_1, \dots, E_k\}$  and capacities  $c_e$  on edges, the load  $\ell_e^\mathcal{E}$  on an edge  $e$  is defined as the number of cuts that contain  $e$ , that is,  $\ell_e^\mathcal{E} = |\{E_i \in \mathcal{E} | e \in E_i\}|$ . Likewise, for a collection of vertex partitions  $\mathcal{C} = \{C_1, \dots, C_k\}$ ,  $\ell_e^\mathcal{C} = |\{C_i \in \mathcal{C} | e \in \delta(C_i)\}|$ .

The input to a *multiway cut packing* problem (MCP) is a graph  $G = (V, E)$  with non-zero integral capacities  $c_e$  on edges, and  $k$  sets  $S_1, \dots, S_k$  of terminals (called “commodities”); each terminal  $i \in S_a$  resides at a vertex

although an optimal solution may end up separating  $u$  from  $w$  in some of the cuts. We are not aware of any other approximation preserving reduction between the two problems.

$r_i$  in  $V$ . The goal is to produce a collection of cuts  $\mathcal{E} = \{E_1, \dots, E_k\}$ , such that (1)  $\forall a \in [k]$ , and  $\forall i, j \in S_a$ ,  $i \neq j$ , the cut  $E_a$  separates  $r_i$  and  $r_j$ , and (2) the maximum “relative load” on any edge,  $\max_e \ell_e^{\mathcal{E}}/c_e$ , is minimized.

The multiway cut packing is NP-hard to solve optimally [1] and we present an LP-rounding based approximation algorithm for it. We assume without loss of generality that the optimal solution has a relative load of 1. The integer program **MCP-IP** below encodes the set of solutions to the MCP with relative load 1. We relax the final constraint to  $d_a(e) \in [0, 1]$  for all  $a \in [k]$  and  $e \in E$  to obtain the linear program **MCP-LP**. It is immediate that if an instance of the MCP admits a solution with relative load 1, then the feasible region of **MCP-LP** is non-empty.

|  |                                    |
|--|------------------------------------|
| <b>(MCP-IP)</b>                        |                                    |
| $d_a(r_i, r_j) \geq 1$                 | $\forall a \in [k], i, j \in S_a$  |
| $d_a(u, v) \leq d_a(u, w) + d_a(w, v)$ | $\forall a \in [k], u, v, w \in V$ |
| $\sum_a d_a(e) \leq c_e$               | $\forall e \in E$                  |
| $d_a(e) \in \{0, 1\}$                  | $\forall a \in [k], e \in E$       |

In the sequel we focus exclusively on solutions to the MCP that are collections of vertex partitions. This is without loss of generality (up to a factor of 2 in edge loads) and allows us to exploit structural properties of vertex sets such as laminarity that help in constructing a good approximation. Accordingly, in the rest of the paper we use the term “cut” to denote a subset of the vertices that defines a vertex partition.

A pair of cuts  $C_1, C_2 \subset V$  is said to “cross” if all of the sets  $C_1 \cap C_2$ ,  $C_1 \setminus C_2$ , and  $C_2 \setminus C_1$  are non-empty. A collection of cuts is said to be *laminar* if no pair of cuts in the collection crosses. Our algorithm is based on the observation that the MCP admits near-optimal laminar solutions. Specifically, there is a polynomial-time algorithm that given a fractional feasible solution to MCP (i.e. a feasible solution to **MCP-LP**) produces a laminar family of fractional cuts that is feasible for the given instance and has small load. This is formalized in Lemma 2.1 below. We first formally define a fractional laminar family of cuts.

**DEFINITION 2.1.** *Given an instance of the MCP on a graph  $G$  with edge capacities  $c_e$  and commodities  $S_1, \dots, S_k$ , a collection of cuts  $\mathcal{C}$  with weight function  $w$  is said to be a fractional laminar cut family feasible for the instance if it satisfies the following properties:*

1. *The collection is laminar.*
2. *Each cut  $C$  in the family is associated with a unique terminal in  $T = \cup_{a \in [k]} S_a$ . We use  $\mathcal{C}_i$  to denote the sub-collection of cuts associated with terminal  $i \in T$ . Every  $C \in \mathcal{C}_i$  contains the node  $r_i$ , and, the total weight of cuts in  $\mathcal{C}_i$ ,  $\sum_{C \in \mathcal{C}_i} w(C)$ , is 1.*

3. *For all  $a \in [k]$  and  $i, j \in S_a$ ,  $i \neq j$ , either  $r_j \notin \cup_{C \in \mathcal{C}_i} C$ , or  $r_i \notin \cup_{C \in \mathcal{C}_j} C$ .*

4. *For every edge  $e \in E$ ,  $\ell_e^{\mathcal{C}} \leq c_e$ .*

Conditions 2 and 3 above ensure feasibility and separation, and condition 4 enforces capacity constraints. Note that for a terminal pair  $i \neq j$  belonging to the same commodity, condition 3 is weaker than requiring cuts in *both*  $\mathcal{C}_i$  and  $\mathcal{C}_j$  to separate  $r_i$  from  $r_j$ .

**LEMMA 2.1.** *Consider an instance of the MCP with graph  $G = (V, E)$ , edge capacities  $c_e$ , and commodities  $S_1, \dots, S_k$ . Given a feasible solution  $d$  to **MCP-LP**, algorithm Lam produces a fractional laminar cut family  $\mathcal{C}$  that is feasible for the MCP on  $G$  with edge capacities  $8c_e + o(1)$ .*

Lemma 2.1 is proved in Section 4. In Section 3 we show how to deterministically round a fractional laminar solution to the MCP into an integral one while increasing the load on every edge by no more than a small additive amount. This rounding algorithm is the main contribution of our work, and crucially uses the laminarity of the fractional solution.

**LEMMA 2.2.** *Given a fractional laminar cut family  $\mathcal{C}$  feasible for the MCP on a graph  $G$  with integral edge capacities  $c_e$ , algorithm Round produces an integral family of cuts  $\mathcal{A}$  that is feasible for the MCP on  $G$  with edge capacities  $c_e + 3$ .*

Combining these lemmas we obtain our main theorem.

**THEOREM 2.1.** *There exists a polynomial-time algorithm that given an instance of the MCP with graph  $G = (V, E)$ , edge capacities  $c_e$ , and commodities  $S_1, \dots, S_k$ , produces a family  $\mathcal{A}$  of multiway cuts, one for each commodity, such that for each  $e \in E$ ,  $\ell_e^{\mathcal{A}} \leq 8c_e + 4$ .*

### 3 Rounding fractional laminar cut families

In this section we develop an algorithm for rounding feasible fractional laminar solutions to the MCP to integral ones while increasing edge loads by a small additive amount. We assume that the edge capacities  $c_e$  are integral.

Our algorithm picks terminals according to an order suggested by the fractional solution and assigns the smallest cuts possible to them subject to the availability of capacity on the edges. At every step we modify the remaining fractional solution to ensure that it continues to be feasible for the unassigned terminals and has small edge loads. We use  $\mathcal{C}$  to denote the fractional laminar cut family that we start out with and  $\mathcal{A}$  to denote the integral family that we construct. Recall that for an edge  $e \in E$ ,  $\ell_e^{\mathcal{C}}$  denotes the the load of the fractional cut family  $\mathcal{C}$  on  $e$ , or the “fractional edge load”, and  $\ell_e^{\mathcal{A}}$  denotes the integral edge load—the load of the integral cut family  $\mathcal{A}$  on  $e$ . Let  $T = \cup_a S_a$ .

---

**Input:** Graph  $G = (V, E)$  with capacities  $c_e$  on edges, a set of terminals  $T$  with a fractional laminar cut family  $\mathcal{C}$ .  
**Output:** A collection of cuts  $\mathcal{A}$ , one for each terminal in  $T$ .

---

1. Preprocess the family  $\mathcal{C}$  so that it satisfies the inclusion invariant.
  2. Initialize  $T' = T$ ,  $\mathcal{A} = \emptyset$ ,  $Y, Z = \emptyset$ , and  $M(v) = \{v\}$  for all  $v \in V$ .
  3. While there are terminals in  $T'$  do:
    - (a) Consider the set of unassigned terminals with the maximum depth, and of these let  $i \in T'$  be a terminal that is undominated in the cut inclusion ordering. Let  $E_i = Y \cap \delta(M(r_i))$ .
    - (b) If  $E_i = \emptyset$ , let  $A_i = M(r_i)$ .
    - (c) If  $E_i \neq \emptyset$  (we say that the terminal has “defaulted” on edges in  $E_i$ ), let  $U_i$  denote the set of end-points of edges in  $E_i$  that lie in  $M(r_i)$ . If  $r_i \in U_i$ , abort and return error. Otherwise, consider the vertex in  $U_i$  that entered  $M(r_i)$  first during the algorithm’s execution, call this vertex  $u_i$ . Set  $A_i$  to be the meta-node of  $r_i$  just prior to the iteration where  $M(u_i)$  becomes equal to  $M(r_i)$ .
    - (d) Add  $A_i$  to  $\mathcal{A}$ . Remove  $\mathcal{C}_i$  from  $\mathcal{C}$  and  $i$  from  $T'$ . For every  $j \in T'$  and  $C \in K_{r_i}^1 \cap \mathcal{C}_j$ , let  $C = C \setminus \{M(r_i)\}$ .
    - (e) If for some edge  $e$ ,  $\ell_e^A = c_e + 2$  and  $\ell_e^C > 0$ , add  $e$  to  $Y$ . If there exists an edge  $e = (u, v)$  with  $\ell_e^C = 0$ , merge the meta-nodes  $M(u)$  and  $M(v)$  (we say that the edge  $e$  has been “contracted”). Add all edges  $e$  with  $\ell_e^C = 0$  to  $Z$  and remove them from  $Y$ .
    - (f) Recompute the depths of vertices and terminals.
- 

Figure 1: Algorithm *Round*—Rounding algorithm for multiway cut packing

We round fractional cuts for terminals roughly in the order of “innermost” terminals first. Specifically, for every vertex  $v \in V$ , let  $K_v$  denote the set of cuts in  $\mathcal{C}$  that contain  $v$ . The “depth” of a vertex  $v$  is the total weight of all cuts in  $K_v$ :  $d_v = \sum_{C \in K_v} w(C)$ . The depth of a terminal is defined as the depth of the vertex at which it resides. Terminals are picked in order of decreasing depth.

When there are two or more terminals of equal maximum depth, we break ties according to the “cut-inclusion” ordering. For a terminal  $i \in T$ , let  $O_i$  denote the largest (outermost) cut in  $\mathcal{C}_i$ , that is,  $\forall C \in \mathcal{C}_i, C \subseteq O_i$ . We say that terminal  $i$  dominates terminal  $j$  in the cut-inclusion ordering, written  $i >_{CI} j$ , if  $O_i \subset O_j$  (if  $O_i = O_j$  we break ties arbitrarily but consistently). Cut-inclusion defines a partial order on terminals. Note that we can pre-process the cut family  $\mathcal{C}$  by reassigning cuts among terminals, such that for all pairs of terminals  $i, j \in T$  with  $i >_{CI} j$ , and all cuts  $C_i \in \mathcal{C}_i$  and  $C_j \in \mathcal{C}_j$  with  $r_i, r_j \in C_i \cap C_j$ , we have  $C_i \subseteq C_j$ . We call this property the “inclusion invariant”. Ensuring this invariant requires a straightforward pairwise reassignment of cuts among the terminals, and we omit the details. Note that following this reassignment, the new outermost cut  $O_i$  of every terminal  $i$  is a subset of or equal to its original outermost cut.

As the algorithm proceeds we modify the cut collection  $\mathcal{C}$ . For example, we may split a cut  $C$  into two cuts containing the same nodes as  $C$  and with weights summing to that of  $C$ . As cuts in  $\mathcal{C}$  are modified, their ownership by terminals remains unchanged, and we therefore continue using the same notation for them. Furthermore, if for two cuts  $C_1$  and  $C_2$ , we have (for example)  $C_1 \subseteq C_2$  at the beginning of the

algorithm, this relationship continues to hold throughout the algorithm. This implies that the inclusion invariant continues to hold throughout the algorithm. We ensure that throughout the execution of the algorithm the cut family  $\mathcal{C}$  continues to be a fractional laminar family for terminals  $T'$  that are yet to be assigned integral cuts. At any point of time, the depth of a vertex or a terminal is defined with respect to the current fractional family  $\mathcal{C}$ .

Before we describe the algorithm we need some more notation. At any point during the algorithm we use  $S_e$  to denote the set of cuts in  $\mathcal{C}$  crossing an edge  $e$  —  $S_e = \{C \in \mathcal{C} | e \in \delta(C)\}$ . Whenever the fractional load of an edge becomes 0, we merge its end-points to form “meta-nodes”. At any point of time, we use  $M(v)$  to denote the meta-node containing a node  $v \in V$ . Finally, for a vertex  $v$ ,  $K_v^1$  denotes the inner-most cuts in  $K_v$  with total weight exactly 1.

The rounding algorithm is given in Figure 1. Roughly speaking, at every step, the algorithm picks a maximum depth terminal  $i$  and assigns the cut  $M(r_i)$  to it (recall that  $M(r_i)$  is the meta-node of the vertex  $r_i$  where terminal  $i$  resides). It “pays” for this cut using fractional cuts in  $K_{r_i}^1$ . Of course some of the cuts in  $K_{r_i}^1$  belong to other commodities, and need to be replaced with new fractional cuts. The cut-inclusion invariant ensures that these other commodities reside at meta-nodes other than  $M(r_i)$ , so we modify each cut in  $K_{r_i}^1 \setminus \mathcal{C}_i$  by removing  $M(r_i)$  from it (see Figure 2). This process potentially increases the total loads on edges incident on  $M(r_i)$  by small amounts, but on no other edges. Step 3c of the algorithm deals with the case in which edges incident on  $M(r_i)$  are already overloaded; In

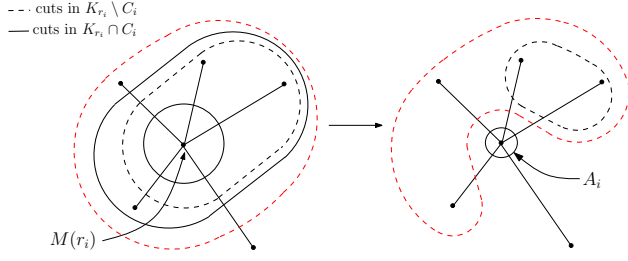


Figure 2: An iteration of Algorithm *Round* (Steps 3b & 3d)

this case we avoid loading those edges further by assigning to  $i$  some subset of the meta-node  $M(r_i)$ . Lemmas 3.4 and 3.5 show that this case does not arise too often.

For a terminal  $i$  and edge  $e$ , if at the time that  $i$  is picked in Step 3a of the algorithm  $e$  is in  $\delta(M(r_i))$ , we say that  $i$  accesses  $e$ . If  $e \in E_i$ , we say that  $i$  defaults on  $e$ , and if  $e$  is in  $\delta(A_i)$  after this iteration, then we say that  $i$  loads  $e$ .

We divide edges into five sets according to their fractional and integral loads. Let  $X_{-1}$  denote the set of edges with  $\ell_e^A \leq c_e - 1$  and  $\ell_e^C > 0$ . For  $a \in \{0, 1\}$ , let  $X_a$  denote the set of edges with  $\ell_e^A = c_e + a$  and  $\ell_e^C > 0$ .  $Y$  denotes the set of edges with  $\ell_e^A \geq c_e + 2$  and  $\ell_e^C > 0$ , and  $Z$  denotes the set of edges with  $\ell_e^C = 0$ . Every edge starts out with a zero integral load. As the algorithm proceeds, the edge goes through one or more of the  $X_a$ s, may enter the set  $Y$ , and eventually ends up in the set  $Z$ . When an edge enters  $Z$ , we merge the end-points of the edge into a single meta-node. When an edge enters  $Y$ , we avoid loading it further (Step 3c), and instead load some edges in  $Z$ . We ensure that edges in  $Z$  are loaded no more than once.

The following lemmas give the desired guarantees on the edges' final loads: Lemmas 3.1 and 3.2 analyze the loads of edges in  $X_a$  for  $a \in \{-1, 0, 1\}$ ; Lemma 3.3 analyzes edges in  $Y$  and Lemmas 3.4 and 3.5 analyze edges in  $Z$ .

**LEMMA 3.1.** *At any point of time, for every edge  $e \in X_{-1}$ ,  $\ell_e^A + \ell_e^C \leq c_e$ .*

*Proof.* We prove the claim by induction over time. Note that at the beginning of the algorithm, we have for all edges  $\ell_e^C \leq c_e$  and  $\ell_e^A = 0$ , so the inequality  $\ell_e^A + \ell_e^C \leq c_e$  holds.

Let us now consider a single iteration of the algorithm and suppose that the edge  $e$  remains in the set  $X_{-1}$  after this step. There are three events that influence the load of the edge  $e = (u, v)$ : (1) a terminal at some vertex in  $M(u)$  accesses  $e$ ; (2) a terminal at  $M(v)$  accesses  $e$ ; and, (3) a terminal at some other meta-node  $M \neq M(u), M(v)$  is assigned an integral cut. Let us consider the third case first, and suppose that a terminal  $i$  is assigned. Since  $A_i \subseteq M$  and therefore  $e \notin \delta(A_i)$  its integral load does not increase. However, in the event that  $S_e \cap C_i$  is non-empty, the fractional load on  $e$  may decrease (because cuts in  $C_i$  are removed from  $C$ ). Therefore, the inequality continues to hold.

Next we consider the case where a terminal  $i$  with  $r_i \in M(u)$  accesses  $e$  (the second case is similar). Then  $M(r_i) = M(u)$ . In this case the integral load of  $e$  potentially increases by 1 (if  $i$  loads the edge). By the definition of  $X_{-1}$ , the new integral load on this edge is no more than  $c_e - 1$ . The fractional load on  $e$  changes in three ways:

- Cuts in  $C_i \cap S_e$  are removed from  $C$ , decreasing  $\ell_e^C$ .
- Some of the cuts in  $(K_{r_i}^1 \setminus C_i) \setminus S_e$  get “shifted” on to  $e$  increasing  $\ell_e^C$  (we remove the meta-node  $M(r_i)$  from these cuts, and they may continue to contain  $M(v)$ ).
- Cuts in  $(K_{r_i}^1 \setminus C_i) \cap S_e$  get shifted off from  $e$  decreasing  $\ell_e^C$  (these cuts initially contain  $M(r_i)$  but not  $M(v)$ , and during this step we remove  $M(r_i)$  from these cuts).

So the decrease in  $\ell_e^C$  is at least the total weight of  $K_{r_i}^1 \cap S_e$ , whereas the increase is at most the total weight of  $K_{r_i}^1 \setminus S_e$ .

In order to account for the two terms, let  $\alpha$  denote the total weight of cuts in  $K_u \cap S_e$ , and  $\beta$  denote the total weight of cuts in  $K_v \cap S_e$ . Then,  $\alpha + \beta = \ell_e^C$ . Moreover, all cuts in  $C \setminus S_e$  either contain both or neither of  $u$  and  $v$ . So we can relate the depths of  $v$  and  $u$  in the following way:  $d_v = d_u - \alpha + \beta$ . Since  $i$  is the terminal picked during this iteration, we must have  $d_u \geq d_v$ , and therefore,  $\alpha \geq \beta$ .

Now, suppose that  $\alpha \geq 1$ . Then  $K_u^1 \subseteq S_e$ . Therefore, the decrease in  $\ell_e^C$  due to the sets  $K_u^1 \cap S_e = K_u^1$  is at least 1, and there is no corresponding increase, so the sum  $\ell_e^A + \ell_e^C$  remains at most  $c_e$ .

Finally, suppose that  $\alpha < 1$ . Then  $K_u^1$  contains all the cuts in  $K_u \cap S_e$ , the weight of  $K_u^1 \cap S_e$  is exactly  $\alpha$ , and so the decrease in  $\ell_e^C$  is at least  $\alpha$ . Moreover, the total weight of  $K_u^1 \setminus S_e$  is  $1 - \alpha$ , therefore, the increase in  $\ell_e^C$  due to the sets in  $K_u^1 \setminus S_e$  is at most  $1 - \alpha$ . Since  $\ell_e^C$  starts out as being equal to  $\alpha + \beta$ , its final value after this step is  $1 - \alpha + \beta \leq 1$  as  $\beta \leq \alpha$ . Noting that  $\ell_e^A$  is at most  $c_e - 1$  after the step, we get the desired inequality.

**LEMMA 3.2.** *For any edge  $e = (u, v)$ , from the time that  $e$  enters  $X_0$  to the time that it exits  $X_1$ ,  $\ell_e^C \leq 1$ . Furthermore suppose (without loss of generality) that during this time in some iteration  $e$  is accessed by a terminal  $i$  with  $r_i \in M(u)$ , then following this iteration until the next time that  $e$  is accessed, we have  $S_e \cap K_u = \emptyset$ , and the next access to  $e$  (if any) is from a terminal in  $M(v)$ .*

*Proof.* First we note that if the lemma holds the first time an edge  $e = (u, v)$  enters a set  $X_a$ ,  $a \in \{0, 1\}$ , then it continues to hold while the edge remains in  $X_a$ . This is because during this time the integral load on the edge does not increase, and therefore throughout this time we assign integral cuts to terminals at meta-nodes different from  $M(u)$  and  $M(v)$  — this only reduces the fractional load on the edge  $e$  and shrinks the set  $S_e$ .

Consider the first time that an edge  $e = (u, v)$  moves from the set  $X_{-1}$  to  $X_0$ . Suppose that at this step we assign an integral cut to a terminal  $i$  residing at node  $r_i \in M(u)$ . Prior to this step,  $\ell_e^A = c_e - 1$ , and so by Lemma 3.1,  $\ell_e^C \leq 1$ . As before define  $\alpha$  to be the total weight of cuts  $K_u \cap S_e$ , and  $\beta$  to be the total weight of cuts  $K_v \cap S_e$ . Then following the same argument as in the proof of Lemma 3.1, we conclude that the final fractional weight on  $e$  is at most  $\beta + 1 - \alpha \leq 1$ . Furthermore, since  $K_u \cap S_e \subseteq K_u^1$ , we either remove all these cuts from  $\mathcal{C}$  or shift them off of edge  $e$ . Moreover, any new cuts that we shift on to  $e$  do not contain the meta-node  $M(r_i) = M(u)$ , and in particular do not contain the vertex  $u$ . Therefore at the end of this step,  $S_e \cap K_u = \emptyset$ . This also implies that following this iteration terminals in  $M(v)$  have depth larger than terminals in  $M(u)$ , and so the next access to  $e$  must be from a terminal in  $M(v)$ .

The same argument works when an edge moves from  $X_0$  to  $X_1$ . We again make use of the fact that prior to the step the fractional load on the edge is at most 1.

**LEMMA 3.3.** *During any iteration of the algorithm, for any edge  $e \in Y$ , the following are satisfied:*

- $\ell_e^C \leq 1$
- *If the edge  $e = (u, v)$  is accessed by a terminal  $i$  with  $r_i \in M(u)$ , then following this iteration until the next time that  $e$  is accessed, we have  $S_e \cap K_u = \emptyset$ , and the next access to  $e$  (if any) is from a terminal in  $M(v)$ .*
- *If a terminal  $i$  with  $r_i \in M(u)$  accesses  $e = (u, v)$ , then  $r_i \neq u$  and  $i$  does not load  $e$ . Also, consider any previous access to the edge by a terminal in  $M(u)$ ; then prior to this access,  $r_i \notin M(u)$ .*

*Proof.* The first two parts of this lemma extend Lemma 3.2 to the case of  $e \in Y$ , and are otherwise identical to that lemma. We omit the details.

For the third part of the lemma, since  $A_i \subseteq M(r_i) = M(u)$  and  $v \notin M(u)$ , we have  $v \notin A_i$ . Next we show that  $u \notin A_i$ , implying that  $i$  does not load the edge  $e = (u, v)$ . Consider the iterations of the algorithm during which  $\ell_e^C \leq 1$ . During this time the edge was accessed at least twice prior to being accessed by  $i$  (once when  $e$  moved from  $X_0$  to  $X_1$ , once when  $e$  moved from  $X_1$  to  $Y$ , and possibly multiple times while  $e \in Y$ ). Let the last two accesses be by the terminals  $j_1$  and  $j_2$ , at iterations  $t_1$  and  $t_2$ ,  $t_1 \leq t_2$ . For  $a \in \{1, 2\}$ , let  $M^a(u)$  and  $M^a(v)$  denote the meta-nodes of  $u$  and  $v$  respectively just prior to iteration  $t_a$ , and  $M(u)$  and  $M(v)$  denote the respective meta-nodes just prior to the current iteration. Then by Lemma 3.2 and the second part of this lemma, we have  $r_{j_1} \in M^1(u)$  and  $r_{j_2} \in M^2(v)$ . We claim that  $i \succ_{CI} j_2 \succ_{CI} j_1$ . Given this claim, if  $r_i \in M^1(u) = M^1(r_{j_1})$ , then since  $i$  and  $j_1$  have the same depth at iteration  $t_1$ , we get a contradiction to the fact that the

algorithm picks  $j_1$  before  $i$  in Step 3a. Therefore,  $r_i \notin M(u)$  at any iteration prior to  $t_1$ , and in particular,  $r_i \neq u$ . Finally, since  $u \in U_i$  and  $U_i \cap A_i = \emptyset$ , this also implies that  $u \notin A_i$ .

It remains to prove the claim. We will prove that  $j_2 \succ_{CI} j_1$ . The proof for  $i \succ_{CI} j_2$  is analogous. In fact we will prove a stronger statement: between iterations  $t_1$  and  $t_2$ , all terminals with cuts in  $S_e$  dominate  $j_1$  in the cut-inclusion ordering. We prove this by induction. By Lemma 3.2, prior to iteration  $t_1$ ,  $S_e$  does not contain any cuts belonging to terminals at  $M(v)$ . Following the iteration,  $S_e$  only contains fractional cuts in  $K_u^1$  that got shifted on to the edge  $e$ . Prior to shifting, these cuts contain  $M^1(u)$ , and therefore  $r_{j_1}$ , but do not belong to  $j_1$ . Then, these cuts are subsets of  $O_{j_1}$ , and so by the inclusion invariant, they belong to terminals dominating  $j_1$  in the cut-inclusion ordering. Therefore, the claim holds right after the iteration  $t_1$ . Finally, following the iteration until the next time that  $e$  is accessed (by  $j_2$ ), the set  $S_e$  only shrinks, and so the claim continues to hold.

In order to analyze the loading of edges in  $Z$ , we need some more notation. Let  $\mathcal{M}$  denote the collection of sets of vertices that were meta-nodes at some point during the algorithm. For any edge  $e \in Z$ , let  $M_e$  denote the meta-node formed when  $e$  enters  $Z$ ; then  $M_e$  is the smallest set in  $\mathcal{M}$  containing both the end points of  $e$ . Note that the collection  $\mathcal{A} \cup \mathcal{M}$  is laminar.

**LEMMA 3.4.** *An edge  $e \in Z$  is loaded only if after the formation of  $M_e$  a terminal residing at a vertex in  $M_e$  defaults on an edge in  $\delta(M_e)$ . (Note that this may happen after  $M_e$  has merged with some other meta-nodes.)*

*Proof.* Let  $i$  be a defaulting terminal that loads the edge  $e \in Z$ . Then  $e \in \delta(A_i)$ , and therefore,  $A_i \subsetneq M_e$  and  $r_i \in M_e$ . Furthermore, since  $A_i$  is a strict subset of  $M_e$ ,  $U_i \cap M_e \neq \emptyset$ , and therefore,  $i$  defaults on an edge  $e' \in Y$  with at least one end-point in  $M_e$ . But if both the end-points of  $e'$  are in  $M_e$ , then we must have  $\ell_{e'}^C = 0$  contradicting the fact that  $e'$  is in  $Y$ . Therefore,  $e' \in \delta(M_e)$ .

**LEMMA 3.5.** *For any meta-node  $M \in \mathcal{M}$ , after its formation, at most one terminal residing at a vertex in  $M$  can default on edges in  $\delta(M)$  (even after  $M$  has merged with other meta-nodes).*

*Proof.* Suppose that two terminals  $i$  and  $j$ , both residing at vertices in  $M$  default on edges in  $\delta(M)$  after the formation of  $M$ , with  $i$  defaulting before  $j$ . Let  $M_1$  ( $M_2$ ) denote the meta-node containing  $M$  just before  $i$  ( $j$ ) defaulted. Note that  $M \subseteq M_1 \subseteq M_2$ . Consider an edge  $e \in E_j \cap \delta(M)$  (recall that  $E_j$  is the set of edges that  $j$  defaults on, so this set is non-empty by our assumption). Then  $e \in \delta(M) \cap \delta(M_2) \subseteq \delta(M_1)$ . Therefore, at the time that  $i$  defaulted,  $e$  was accessed by  $i$ , and by the third claim in Lemma 3.3,  $r_j \notin M_1$ . This contradicts the fact that  $r_j \in M$ .

Finally we can put all these lemmas together to prove our main result on algorithm *Round*.

*Proof of Lemma 2.2:* We first note that the third part of Lemma 3.3 implies that for all  $i$ ,  $r_i \notin U_i$ , and so the algorithm never aborts. Next we claim that  $r_i \in A_i \subseteq O_i$  for all  $i$ , so we obtain a feasible cut packing: each cut  $A_i$  is set equal to the meta-node of  $r_i$  at some stage of the algorithm, and furthermore, at the time that  $i$  is assigned an integral cut,  $A_i \subseteq M(r_i) \subseteq O_i$ . Finally, note that every edge starts out in the set  $X_{-1}$ , goes through one or more of the  $X_a$ 's,  $a \in \{0, 1\}$ , potentially goes through  $Y$  (when its integral load becomes  $c_e + 2$ ), and ends up in  $Z$ . Lemma 3.3 implies that edges in  $Y$  never get loaded, and so when an edge  $e$  enters  $Z$ ,  $\ell_e^A \leq c_e + 2$ . After this point the edge stays in  $Z$ , and Lemmas 3.4 and 3.5 imply that it gets loaded at most once. Therefore, the final load on the edge is at most  $c_e + 3$ .

#### 4 Constructing fractional laminar cut families

We now show that fractional solutions to the program **MCP-LP** can be converted in polynomial time into fractional laminar cut families while losing only a small factor in edge load. For ease of description we interpret cuts as sets of vertices as well as sets of terminals residing at those vertices.

We first show how to convert a feasible *integral* collection of cuts into a feasible integral laminar collection of cuts, losing a factor of 2 in edge loads (see Subroutine *Integer-Lam* in Figure 4, and Lemma 4.1). Obtaining laminarity for an arbitrary fractional solution requires converting it first into an integral solution for a related instance and then applying Lemma 4.1 (see Algorithm *Lam*).

**LEMMA 4.1.** *Consider an instance of the MCP with graph  $G = (V, E)$  and commodities  $S_1, \dots, S_k$ , and let  $\mathcal{C}^1 = \{C_i^1\}_{i \in S_a, a \in [k]}$  be a family of cuts such that for each  $a \in [k]$  and  $i \in S_a$ ,  $C_i^1$  contains  $i$  but no other  $j \in S_a$ . Then Subroutine *Integer-Lam* in Figure 4 produces a laminar cut collection  $\mathcal{C}^2 = \{C_i^2\}_{i \in S_a, a \in [k]}$  such that for each  $a \in [k]$  and  $i \neq j \in S_a$ , either  $C_i^2$  or  $C_j^2$  separates  $i$  from  $j$ , and  $\ell_e^{\mathcal{C}^2} \leq 2\ell_e^{\mathcal{C}^1}$  for every edge  $e \in E$ .*

The subroutine starts by applying a series of simple rules to resolve crossings while maintaining the invariant that pairs of terminals belonging to the same commodity are always separated by at least one of the two cuts assigned to them (see Steps 1 & 2 and Figure 3). Certain kinds of crossings cannot be resolved while maintaining this invariant. In Step 3 we ignore the commodities that each terminal belongs to and assign new laminar cuts to terminals that are subsets of their original cuts (therefore separation continues to be maintained). This step incurs a penalty of 2 in edge loads.

The rough idea behind Step 3 is to consider the set of all ‘‘conflicting’’ terminals, say  $F$ , and to assign to each terminal  $i \in F$  the cut  $\bigcap_{j \in F} \hat{C}_j$  where  $\hat{C}_j$  is either the cut of

terminal  $j$  or its complement depending on which of the two contains  $r_i$ . These intersections are clearly laminar, and are subsets of the terminals’ original cuts. Furthermore, if each terminal gets a unique intersection, then edge loads increase by a factor of at most 2. Unfortunately, some groups of terminals may share the same intersections. To avoid this, we pick terminals in an order suggested by the structure of the conflict graph on terminals (graph  $\mathcal{G}$  in the algorithm) and assign appropriate intersections to them while explicitly enforcing at most a factor of 2 increase in edge loads.

Throughout the algorithm every terminal in  $\cup_a S_a$  has an integral cut assigned to it. The proof of Lemma 4.1 is established in three parts: Lemma 4.2 establishes the laminarity of the output cut family, Lemma 4.4 argues separation, and Lemma 4.5 analyzes edge loads.

**LEMMA 4.2.** *Subroutine *Integer-Lam* runs in polynomial time and produces a laminar cut collection.*

*Proof.* Define the crossing number of a family of cuts to be the number of pairs of cuts that cross each other. Note first that in every iteration of Steps 1 and 2 of the algorithm, the crossing number of the cut family  $\mathcal{C}$  strictly decreases: no new crossings are created during these steps, while the crossings of the two or more cuts involved in each transformation are resolved (see Figure 3). So after a polynomial number of steps, we exit Steps 1 and 2 and go to Step 3.

Next, we claim that during Step 3 of the algorithm the graph  $\mathcal{G}$  is acyclic. This implies that while  $\mathcal{G}$  is non-empty, we can always find a leaf terminal in Step 3; therefore every terminal in  $\mathcal{G}$  gets assigned a new cut. It is immediate that the graph does not contain any directed blue cycles or any directed red cycles (the latter follows because red edges define a partial order over terminals). Suppose the graph contains three terminals  $i_1, i_2$  and  $i_3$  with a red edge from  $i_1$  to  $i_2$ , and a red or blue edge from  $i_2$  to  $i_3$ , then it is easy to see that there must be a red or blue edge from  $i_1$  to  $i_3$ . Therefore, any multi-colored directed cycle must reduce to either a smaller blue cycle or a cycle of length 2. Neither of these is possible (the latter is ruled out by definition).

Now consider cuts assigned during Step 3. Let  $T$  be the set of terminals corresponding to some component in  $\mathcal{G}$  and  $j \notin T$ . Then before  $T$  is processed,  $j$ 's cut is laminar with respect to all the cuts in  $A_T$ , and is therefore a subset of some meta-node in  $G_{pT}$ . So the new cuts assigned to terminals in  $T$  are also laminar with respect to  $j$ 's cut.

Finally, consider cuts assigned during Step 3 to terminals in the same component. Consider the set of all meta-nodes created during the processing of this component. This set is laminar, and the new cuts assigned are a subset of this laminar family. Therefore, they are laminar.

**LEMMA 4.3.** *For a commodity  $i$  assigned a cut in Step 3 of the subroutine, let  $C_i^1$  be its cut before this step, and  $C_i^2$  be the new cut assigned to it. Then  $C_i^2 \subseteq C_i^1$ .*

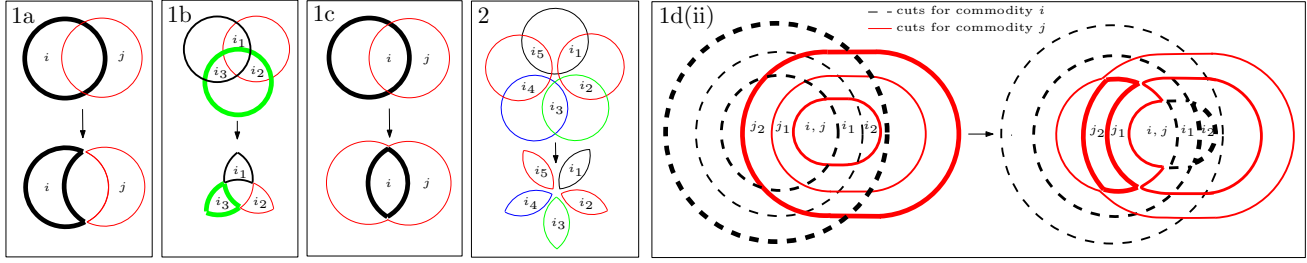


Figure 3: Some simple rules for resolving crossing cuts. See subroutine *Integer-Lam* in Figure 4 for formal descriptions.

*Proof.* We assume without loss of generality that prior to Step 3 each edge load is at most one; this can be achieved by splitting a multiply-loaded edge into many edges. We focus on the behavior of the algorithm for a single component  $T$  of  $\mathcal{G}$  and prove the lemma by induction over time.

Consider an iteration of Step 3 during which some terminal  $i \in T$  is assigned and let  $C_i$  be its original cut. Consider any vertex  $v \notin C_i$  and let  $P$  be a shortest simple path from  $r_i$  to  $v$  in  $G_{p^T}$  (where the length of an edge  $e$  is given by  $p_e^T$  just prior to when  $i$  is assigned a new cut). It is easy to see that there is one such shortest path that crosses each new cut assigned prior to this iteration in Step 3 at most twice – suppose there are multiple entries and exits for some cut, we can “short-cut” the path by connecting the first point on the path inside the cut to the last point on the path inside the cut via a simple path of length 0 lying entirely inside the cut. We pick  $P$  to be such a path. We will prove that  $P$ ’s length is at least 2. So the meta-node containing  $i$  must lie inside the cut  $C_i$ , and the lemma holds.

Let  $T_1$  (resp.  $T_2$ ) be the set of terminals in  $T \setminus C_i$  (resp.  $T \cap C_i$ ) that are assigned new cuts before  $i$  in this iteration. We first note that for any  $j$  in  $T_1$ , prior to this step, there is no edge from  $j$  to  $i$  (as  $j$  is assigned before  $i$ ), so  $r_i \notin C_j$ , and this along with  $r_j \notin C_i$  implies that  $C_i$  and  $C_j$  are disjoint. This implies that the new cut of  $j$  (which is a subset of  $C_j$  by induction) is also disjoint from  $C_i$ , and therefore cannot load any edge with an end-point in  $C_i$ . So the only new cuts assigned this far in Step 3 that load edges in  $P$  belong to terminals in  $T_2$ .

Now we will analyze  $P$ ’s length by accounting for all the newly assigned cuts that load its edges. Let  $S_P$  be the set of terminals in  $T_2$  that load an edge in  $P$ , and  $j \in S_P$ . Since the new cut of  $j$  intersects  $P$ , by the induction hypothesis,  $C_j$  should either intersect  $P$  or contain the entire path inside it. If  $C_j$  contains  $P$  entirely, then  $C_j \setminus C_i \neq \emptyset$ , and furthermore  $r_i, r_j \in C_i \cap C_j$ . This implies that either  $C_i \subset C_j$  and there is a directed red edge from  $j$  to  $i$ , or  $C_i \setminus C_j \neq \emptyset$ , that is,  $C_i$  and  $C_j$  cross and should have matched the rule in Step 1d of the algorithm. Both possibilities lead to a contradiction. Therefore,  $C_j$  must intersect  $P$ .

Finally, the original total length of the path is at least  $2|S_P| + 2$ , because each terminal in  $S_P$  contributes two units

towards its length, and another two units is contributed by  $C_i$ . Out of these up to  $2|S_P|$  units of length is consumed by terminals in  $S_P$ . Therefore, at the time that  $i$  is assigned a cut, at least 2 units remain.

LEMMA 4.4. *When subroutine Integer-Lam terminates,  $\forall a \in [k]$  and  $i \neq j \in S_a$ , either  $C_i$  or  $C_j$  separates  $i$  and  $j$ .*

*Proof.* We claim that for every  $a \in [k]$  and  $i \neq j \in S_a$ , at every time step during the execution of the algorithm,  $|C_i \cap C_j \cap \{r_i, r_j\}| \leq 1$ . Then since by Lemma 4.2 the final solution is laminar, the lemma follows. We prove this claim by induction over time. First, if during any iteration we “shrink” the cut of any terminal (that is, reassign to the terminal a cut that is a strict subset of its original cut), then the claim continues to hold for that terminal, because intersections of the terminal’s cut only shrink in that step. Note that cuts of terminals expand only in Steps 1c and 1d of the algorithm (by construction and by Lemma 4.3).

Suppose that during some iteration we apply the transformation in Step 1c to terminals  $i$  and  $j$ , reassigning  $C_j = C_i \cup C_j$ , and the claim fails to hold for terminal  $j$ . Specifically, suppose that for some  $j' \in S_a$ , after the iteration we have  $r_j, r_{j'} \in C_j \cap C_{j'}$ . Then,  $r_j \in C_{j'}$ , and therefore  $C_{j'}$  intersected  $C_j$  prior to the iteration, and by the induction hypothesis  $r_{j'} \in C_i \setminus C_j$  prior to the iteration. If  $r_i \in C_{j'}$ , then prior to the iteration,  $i$  and  $j'$  contradicted the induction hypothesis. Otherwise,  $i, j$  and  $j'$  satisfy the conditions in Step 1b of the algorithm, and this contradicts the fact that we apply the transformation in Step 1c at this iteration. A similar argument applies to the rule in the first part of Step 1d.

Finally, suppose that during some iteration we apply the transformation in the second part of Step 1d. Then the cut assigned to every  $i_{x'}$  for  $x' \leq x-2$  is a subset of the previous cut of  $i_{x'+1}$  but does not contain  $i_{x'+1}$ ; the cut assigned to  $i_x$  is a subset of its original cut;  $i_{x-1}$  does not belong to any of the new cuts except its own. So once again arguing as before, the induction hypothesis continues to hold for these terminals. The same argument holds for the  $j_{y'}$  terminals.

LEMMA 4.5. *Let  $C^1$  be the cut collection input to Subroutine Integer-Lam, and  $C^2$  the cut collection output by it. Then  $\ell_e^{C^2} \leq 2\ell_e^{C^1}$ .*

---

**Input:** Graph  $G = (V, E)$  with edge capacities  $c_e$ , commodities  $S_1, \dots, S_k$ , a feasible solution  $d$  to the program **MCP-LP**.  
**Output:** A fractional laminar family of cuts  $\mathcal{C}$  that is feasible for  $G$  with edge capacities  $8c_e + o(1)$ .

---

Subroutine *Integer-Lam* (**Input:** An integral cut family  $\mathcal{C}$ . **Output:** An integral laminar cut family.)

1. While there are pairs of cuts in  $\mathcal{C}$  that cross, do (see also Figure 3):
  - (a) Consider any pair of crossing cuts  $C_i, C_j \in \mathcal{C}$  belonging to terminals  $i \neq j$ , such that  $r_i \in C_i \setminus C_j$  and  $r_j \in C_j \setminus C_i$ . Reassign  $C_i = C_i \setminus C_j$  and  $C_j = C_j \setminus C_i$ . Return to Step 1.
  - (b) Consider any three terminals  $i_1, i_2, i_3$  with cuts  $C_1, C_2$  and  $C_3$  such that  $r_{i_1} \in C_1 \cap C_2 \setminus C_3$ ,  $r_{i_2} \in C_2 \cap C_3 \setminus C_1$ , and  $r_{i_3} \in C_3 \cap C_1 \setminus C_2$ . Then, reassign these respective intersections to the three terminals. Return to Step 1.
  - (c) Consider any pair of crossing cuts  $C_i, C_j \in \mathcal{C}$  belonging to terminals  $i, j \in S_a$  for some  $a$ , such that  $r_i \in C_i \cap C_j$  and  $r_j \in C_j \setminus C_i$ . Reassign  $C_i = C_i \cap C_j$  and  $C_j = C_i \cup C_j$ . Return to Step 1.
  - (d) Consider any pair of crossing cuts  $C_i, C_j \in \mathcal{C}$  belonging to  $i \neq j$ , such that  $r_i, r_j \in C_i \cap C_j$ ,  $i \in S_a$ , and  $j \in S_b$  with  $a \neq b$ .
    - If there is no  $i' \in S_a \cap C_j$  with  $C_i \subset C_{i'}$ , reassign  $C_i = C_i \cup C_j$  and  $C_j = C_i \cap C_j$ . Conversely, if there is no  $j' \in S_b \cap C_i$  with  $C_j \subset C_{j'}$ , reassign  $C_j = C_i \cup C_j$  and  $C_i = C_i \cap C_j$ . (This is similar to Step 1c.) Return to Step 1.
    - If neither of those cases hold, let  $i_0 = i$ , and let  $i_1, \dots, i_x$  denote the terminals in  $S_a \cap C_j$  with  $C_i \subset C_{i_1} \subset C_{i_2} \subset \dots \subset C_{i_x}$ . For  $x' \leq x - 2$ , reassign  $C_{i_{x'}} = (C_{i_{x'+1}} \setminus C_j) \cup C_{i_{x'}}$ ,  $C_{i_{x-1}} = C_{i_x} \cup C_j$ , and  $C_{i_x} = C_{i_x} \cap C_j \setminus C_{i_{x-1}}$ . Reassign cuts to  $j$  and terminals in  $S_b \cap C_i$  likewise. Return to Step 1.
  - (e) If none of the above rules match, then go to Step 2.
2. Let  $\mathcal{G}$  be a directed graph on the vertex set  $\cup_a S_a$ , with edges colored red or blue, defined as follows: for terminals  $i \neq j$ ,  $\mathcal{G}$  contains a red edge from  $i$  to  $j$  if and only if  $C_j \subset C_i$ , and contains a blue edge from  $i$  to  $j$  if and only if  $r_j \in C_i$ ,  $r_i \notin C_j$ , and  $C_j \setminus C_i \neq \emptyset$ . We note that since no pair of terminals  $i$  and  $j$  matches the rules in Step 1, whenever  $C_i$  and  $C_j$  intersect  $\mathcal{G}$  contains an edge between  $i$  and  $j$ . While there is a directed blue cycle in  $\mathcal{G}$ , consider the shortest such cycle  $i_1 \rightarrow i_2 \rightarrow \dots \rightarrow i_x \rightarrow i_1$ . For  $x' \leq x$ ,  $x' \neq 1$ , assign to  $i_{x'}$  the cut  $C_{i_{x'}} \cap C_{i_{x'-1}}$ , and assign to  $i_1$  the cut  $C_{i_1} \cap C_{i_x}$ .
3. We show in Lemma 4.2 that at this step  $\mathcal{G}$  is acyclic. For every connected component in  $\mathcal{G}$  do:

Let  $T$  be the set of terminals in the component and  $A_T$  be the set of corresponding cuts. Assign capacities  $p_e^T = 2\ell_e^{A_T}$  to edges in  $G$ . Let  $G_{p^T}$  be the graph obtained by merging all pairs of vertices that have an edge  $e$  with  $p_e^T = 0$  between them. We call the vertices of  $G_{p^T}$  “meta-nodes” (note that these are sets of vertices in the original graph). At any point of time, let  $R_i$  denote the meta-node at which a terminal  $i$  resides.

While there are terminals in  $T$ , pick any “leaf” terminal  $i$  (that is, a terminal with no outgoing red or blue edges in  $\mathcal{G}$ ). Reassign to  $i$  the cut  $R_i$ . Reduce the capacity of every edge  $e \in \delta(R_i)$  by 1. Remove  $i$  from  $T$ ; remove  $i$  and all edges incident on it from  $\mathcal{G}$ . Recompute the graph  $G_{p^T}$  based on the new capacities.

---

Algorithm *Lam*

1. For every  $a \in [k]$  and  $i \in S_a$  do: Order the vertices in  $G$  in increasing order of their distance under  $d_a$  from  $r_i$ . Let this ordering be  $v_0 = r_i, v_1, \dots, v_n$ . Let  $\mathcal{C}_i^1$  be the collection of cuts  $\{v_0, v_1, \dots, v_b\}$ , one for each  $b \in [n]$  with  $d_a(r_i, v_b) < 0.5$ , with weights  $w^1(\{v_0, \dots, v_b\}) = 2(\min\{d_a(r_i, v_{b+1}), 0.5\} - d_a(r_i, v_b))$ , and  $\mathcal{C}^1 = \{\mathcal{C}_i^1\}_{i \in \cup_a S_a}$ .
  2. Let  $N = n \sum_a |S_a|$ . Round up the weights of all the cuts in  $\mathcal{C}^1$  to multiples of  $1/N^2$ , and truncate the collection so that the total weight of every sub-collection  $\mathcal{C}_i^1$  is exactly 1. Furthermore, split every cut with weight more than  $1/N^2$  into multiple cuts of weight exactly  $1/N^2$ , assigned to the same commodity. Call this new collection  $\mathcal{C}^2$  with weight function  $w^2$ . Note that every cut in this collection has weight exactly  $1/N^2$ .
  3. Construct a new instance of MCP in  $G$  as follows. For each  $a \in [k]$ , construct  $N^2$  new commodities with terminal sets identical to that of  $S_a$  (i.e., the terminals reside at the same nodes). For every new terminal corresponding to an older terminal  $i$ , assign to the new terminal a unique cut from  $\mathcal{C}_i^2$  with weight 1. Call this new collection  $\mathcal{C}^3$ , and the new instance  $I$ .
  4. Apply Subroutine *Integer-Lam* above to the family  $\mathcal{C}^3$  to obtain family  $\mathcal{C}^4$ .
  5. For every  $a \in [k]$  and every  $i \in S_a$ , let  $\mathcal{C}_i^5$  be the set of  $N^2/2$  innermost cuts in  $\mathcal{C}^4$  assigned to terminals in the new instance  $I$  that correspond to terminal  $i$ . (Note that these cuts are concentric as they belong to a laminar family and all contain  $r_i$ . Therefore “innermost” cuts are well defined.) Assign a weight of  $2/N^2$  to every cut in this set. Output the collection  $\mathcal{C}^5$ .
- 

Figure 4: Algorithm *Lam*—Algorithm to convert an LP solution into a feasible fractional laminar family

*Proof.* We first claim that edge loads are preserved throughout Steps 1 and 2. This can be established via a case-by-case analysis by noting that in each step the number of new cuts that an edge crosses is no more than the number of old cuts it crosses prior to the transformation. It remains to analyze Step 3. We claim that we only lose a factor of 2 in edge loads during this step of the algorithm. This is easy to see. Note that for every edge  $e$ ,  $\sum_T p_e^T \leq 2\ell_e^{\mathcal{C} \cup T}$ , where  $\mathcal{C} \cup T$  is the family of cuts belonging to terminals in any non-singleton component of  $\mathcal{G}$  prior to Step 3. Moreover, in each iteration of the step, we only load an edge  $e$  to the extent of  $p_e^T$ .

The proof of Lemma 4.1 follows immediately from Lemmas 4.2, 4.4 and 4.5. Given this lemma, algorithm *Lam* in Figure 4 converts an arbitrary feasible solution for **MCP-LP** into a feasible fractional laminar family.

*Proof of Lemma 2.1:* Note first that the cut collection  $\mathcal{C}^1$  satisfies the following properties: (1) For every  $a \in [k]$  and  $i \in S_a$ , every cut in  $\mathcal{C}_i^1$  contains  $r_i$ , but not  $r_j$  for  $j \in S_a, j \neq i$ ; (2) The total weight of cuts in  $\mathcal{C}_i^1$  is 1; (3) For every edge  $e$ ,  $\ell_e^{\mathcal{C}^1} \leq 2 \sum_a d_a(e) \leq 2c_e$ . The family  $\mathcal{C}^2$  also satisfies the first two properties, however loads the edges slightly more than  $\mathcal{C}^1$ . Any edge belongs to at most  $N$  cuts, and therefore the load on the edge goes up by an additive amount of at most  $1/N$ . Therefore, for every  $e$ ,  $\ell_e^{\mathcal{C}^2} \leq 2c_e + 1/N$ . Next, the collection  $\mathcal{C}^3$  is a feasible integral family of cuts for the new instance  $I$  with  $\ell_e^{\mathcal{C}^3} = N^2 \ell_e^{\mathcal{C}^2}$ . Therefore, applying Lemma 4.1, we get that  $\mathcal{C}^4$  is a feasible laminar integral family of cuts for  $I$  with  $\ell_e^{\mathcal{C}^4} \leq 2N^2 \ell_e^{\mathcal{C}^2}$ . Finally, in family  $\mathcal{C}^5$ , every terminal  $i \in S_a$  gets assigned  $N^2/2$  fractional cuts, each with weight  $2/N^2$ . So the total weight of cuts in  $\mathcal{C}_i^5$  is 1. Now consider any two terminals  $i, j \in S_a, i \neq j$ . Then, in all the  $N^2$  commodities corresponding to  $S_a$  in instance  $I$ , either the cut assigned to  $i$ 's counterpart, or that assigned to  $j$ 's counterpart separates  $i$  from  $j$ . Say that among at least  $N^2/2$  of the commodities in  $I'$ , the cut assigned to  $i$ 's counterpart separates  $i$  from  $j$ . Then, the innermost  $N^2/2$  cuts assigned to  $i$  in  $\mathcal{C}^5$  separate  $i$  from  $j$ . Therefore, the family  $\mathcal{C}^5$  satisfies the first three conditions of feasibility as given in Definition 2.1. Finally, it is easy to see that on every edge  $e$ ,  $\ell_e^{\mathcal{C}^5} \leq 2/N^2 \ell_e^{\mathcal{C}^4} \leq 4(2c_e + 1/N)$ .

## 5 Concluding Remarks

Given that our algorithms rely heavily on the existence of good laminar solutions, a natural question is whether every feasible solution to the MCP can be converted into a laminar one with the same load. Figure 5 shows that this is not true. The figure displays one integral solution to the MCP where the solid edges represent the cut for commodity  $a$ , and the dotted edges represent the cut for commodity  $b$ . It is easy to see that this instance admits no fractional laminar solution with load 1 on every edge.

Is the “laminarity gap” small for the more general set

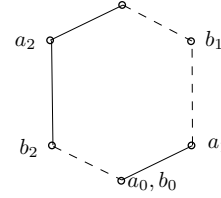


Figure 5: Each edge has capacity 1. There are two commodities with terminal sets  $\{a_0, a_1, a_2\}$  and  $\{b_0, b_1, b_2\}$ .

multiway cut packing and multicut packing problems as well? We believe that this is not the case and there exist instances for both of those problems with a non-constant laminarity gap.

## References

- [1] S. Barman and S. Chawla. Packing multiway cuts in capacitated graphs. <http://arxiv.org/abs/0810.0674>.
- [2] G. Calinescu, H. Karloff, and Y. Rabani. An improved approximation algorithm for multiway cut. *Journal of Computer and System Sciences*, 60(3):564–574, 2000.
- [3] G. Calinescu, H. Karloff, and Y. Rabani. Approximation algorithms for the 0-extension problem. *SIAM Journal on Computing*, 34(2):358–372, 2004.
- [4] C. Chekuri, S. Khanna, J. Naor, and L. Zosin. A linear programming formulation and approximation algorithms for the metric labeling problem. *SIAM J. on Discrete Mathematics*, 18(3):608–625, 2004.
- [5] A. Karzanov. Minimum 0-extensions of graph metrics. *European J. of Combinatorics*, 19(1):71–101, 1998.
- [6] J. Kleinberg and E. Tardos. Approximation algorithms for classification problems with pairwise relationships: metric labeling and Markov random fields. *Journal of the ACM*, 49(5):616–639, 2002.
- [7] M. Li, B. Ma, and L. Wang. On the closest string and substring problems. *Journal of the ACM*, 49(2):157–171, 2002.
- [8] Y. Rabani, L. Schulman, and C. Swamy. Approximation algorithms for labeling hierarchical taxonomies. In *ACM-SIAM Symp. on Discrete Algorithms*, pages 671–680, 2008.
- [9] R. Ravi and J. Kececioglu. Approximation algorithms for multiple sequence alignment under a fixed evolutionary tree. *Discrete Applied Mathematics*, 88:355–366, 1998.
- [10] L. Wang and D. Gusfield. Improved approximation algorithms for tree alignment. *Journal of Algorithms*, 25(2):255–273, 1997.
- [11] L. Wang, T. Jiang, and D. Gusfield. A more efficient approximation scheme for tree alignment. *SIAM Journal on Computing*, 30(1):283–299, 2000.
- [12] L. Wang, T. Jiang, and E. Lawler. Approximation algorithms for tree alignment with a given phylogeny. *Algorithmica*, 16(3):302–315, 1996.