10-1-1994

Hypergraph Partitioning Algorithms

Tom Leighton
Massachusetts Institute of Technology

Fillia Makedon
Dartmouth College

Spyros Tragoudas
Southern Illinois University

HYPERGRAPH PARTITIONING ALGORITHMS

Tom Leighton
Fillia Makedon
Spyros Tragoudas

Technical Report PCS-TR94-233

10/94

Hypergraph Partitioning Algorithms[1]

Tom Leighton
Department of Mathematics & Lab for Computer Science
M.I.T.
Cambridge, MA 02139

Fillia Makedon
Department of Computer Science
Dartmouth College
Hanover, NH 03755

Spyros Tragoudas
Department of Computer Science
Southern Illinois University
Carbondale, IL 62901

Abstract

We present the first polynomial time approximation algorithms for the balanced hypergraph partitioning problem. The approximations are within polylogarithmic factors of the optimal solutions. The choice of algorithm involves a time complexity/approximation bound tradeoff. We employ a two step methodology. First we approximate the flux of the input hypergraph. This involves an approximate solution to a concurrent flow problem on the hypergraph. In the second step, we use the approximate flux to obtain approximations for the balanced bipartitioning problem. Our results extend the approximation algorithms by Leighton–Rao on graphs [21] to hypergraphs. We also give the first polylogarithmic times optimal approximation algorithms for multiway (graph and hypergraph) partitioning problems into bounded size sets. A better approximation algorithm for the latter problem is finally presented for the special case of bounded sets of size at most $O(\log n)$ on planar graphs and hypergraphs, where n is the number of nodes of the input instance.

1 Introduction

A *hypergraph* $G_h = (V_h, E_h)$ consists of n nodes and m *hyperedges*. A hyperedge e in E_h is defined by a subset of nodes V_e in V_h. *Hypergraph partitioning* is the task of dividing G_h into smaller parts. The objective is to partition V_h into sets such that *the complexity of connections* between the nodes in different sets is minimized. A natural way of formalizing the notion of hyperedge complexity is to attribute to each hyperedge in G_h some *connection cost* and to sum the connection costs of all hyperedges connecting nodes in different sets of a partition P. The latter is also referred to as the *cost of the partition* P. For simplicity, we use P to denote both a partition of V_h into sets and the cost of the partition.

Hypergraph partitioning often arises in the context of integrated circuit layout, testing and high-level synthesis in VLSI [18, 1]. A circuit, often given as a *netlist*, is a description of *switching elements* and their connecting *wires*. The elements may have integer weights representing layout area requirements or fabrication costs. Similarly, the wires may have integer costs that express parameters such as the *buswidth* of the net, i.e., the number of bits that can be sent across the net in parallel, or priorities that do not allow critical nets to connect elements in different sets [18]. In *circuit partitioning*, the objective is to partition the elements into sets such that the sum of the weights on the elements in a set is within a prescribed range and the sum of the costs on the nets that connect nodes in more than a set is minimized. The sum of the weights on the elements in the set is also called the *size* of the set.

Circuit partitioning is formalized as an operation on hypergraphs. An *exact* or an *approximation* algorithm for hypergraph partitioning is an exact or an approximation algorithm for circuit partitioning. Thus, the algorithms described in this paper for hypergraph partitioning apply for circuit partitioning as well.

The problem of partitioning a hypergraph with uniform weights on its nodes into sets of size at least three, or into two or more equal size sets, so that the weight of hyperedges connecting nodes in different

[1] Preliminary versions of these results appear in [19, 23].

sets is minimized has been shown to be NP-hard [8]. However, the special case when the sizes of the sets are at most two can be solved polynomially using *graph matching* [8].

The problem of partitioning into two sets whose sizes have a constant ratio (*balanced bipartitioning*) is central in VLSI Layout [18]. *Bipartitioning* algorithms are often used as a basis for partitioning into more than two components. The latter partitioning problem is called *multiway partitioning*, and finds important applications in VLSI layout if the sizes of the components are *balanced*, i.e., satisfy a predefined upper and lower bound. We also refer to this problem as the *balanced multiway partitioning problem* [18]. Variations of the multiway partitioning problem, including the problem of partitioning circuits with uniform weights on the nodes into sets with small constant sizes, find additional applications in Built-In Self-Testing of VLSI circuits [4, 15, 30]. Many *heuristic* solutions, without any provable indication on their performance, have been proposed for *balanced bipartitioning*, *balanced multiway partitioning* [2, 3, 5, 12, 28], and partitioning into sets of small constant size [15, 30].

1.1 Approximation algorithms for partitioning

Little has been done so far in the area of proposing *approximation algorithms* for hypergraph partitioning. The only existing polynomial time approximation algorithms are for bipartitioning problems on graphs. In the latter problem the input is a graph $G = (V, E)$ of n nodes and m edges, and the goal is to partition V into two sets so that the sum of the weights on the edges with endpoints in two different sets is minimized.

Rao [25, 26] proposed approximation algorithms for balanced bipartitions of planar graphs. His method is based on the computation of the *minimum* k-limited quotient separator. Let $l(A)$ denote the size of a set $A \subseteq V$. Rao [25] defined the *k-limited quotient separator* s_k for a partition (A, B) of V as

$$s_k = \frac{\sum_{u \in A, v \in B} w(u, v)}{\min\{l(V)/k, min\{l(A), l(B)\}\}}. \tag{1}$$

The minimum k-limited quotient separator is the minimum among all the k-limited quotient separators of graph G. Rao proved [25, 26] that an approximation to the minimum k-limited quotient node separator s_k suffices to approximate the k-balanced bipartition problem, where we insist that the ratio $l(A)/l(V)$ is at least $1/k$ for a fixed constant k. [2] Let any k and k' such that $k < k'$ and $k \geq 3$, $P(k')$ be the optimal solution for the k'-balanced bipartition problem, and l_{min} be the smallest weight on a node of G.

Theorem 1.1 [By Rao in [26, 25]] *Given a polynomial time algorithm that approximates the minimum k-limited quotient separator within a factor of $K(n)$, we can construct a polynomial time algorithm for the k-balanced bipartition problem with cost $O(K(n) \cdot P(k') \cdot \log(k/(k-k'))$ or an alternative polynomial time algorithm with cost $O(K(n) \cdot \log(l(V)/l_{min}) \cdot P(k))$.*

Let T be the time complexity of the approximation algorithm for the minimum k-limited quotient separator. The time complexity of the first alternative in Theorem 1.1 is $O(n \cdot T)$, and for the second alternative is $O(n \cdot T + n \cdot m \cdot \log n)$. In [25, 26], Rao presents exact algorithms as well as constant times optimal approximations for the minimum k-limited quotient separator on planar graphs.

In addition, Leighton and Rao [21] use *multicommodity flow techniques* to approximate the *flux* or *minimum edge expansion* of a graph $G = (V, E)$ with uniform weights on the nodes within a $O(\log n)$

[2] The k-balanced bipartitioning problem is NP-complete [8].

factor[3]. The multicommodity flow MF problem, which involves many sources and destinations associated to different commodities, is formally defined in Section 1.2. The flux α of graph G is defined as

$$\alpha = \frac{\sum_{u \in A, v \in B} w(u,v)}{\min\{l(A), l(B)\}}. \tag{2}$$

From (1) and (2) we deduce that an $O(\log n)$ approximation for the *flux* leads to an approximation of the minimum k-limited quotient separator for a constant k. Then, by Theorem 1.1, we have:

Theorem 1.2 *Leighton-Rao's algorithm obtains k-balanced bipartitions that have either $O(\log^2 n \cdot P(k))$ or $O(\log n \cdot P(k'))$ cost, where k and k' are constants defined as in Theorem 1.1.*

This algorithm was extended in [20] for graphs with arbitrary weights on the nodes, and obtains $O(\log n)$ approximations of the flux as well as the minimum k-limited quotient separator. The time complexity of the flux approximation algorithms in [21, 20] is determined by the multicommodity flow problems involved in the approximation. In particular, if the instance is a graph with arbitrary weights on the nodes, the expected time complexity is $\tilde{O}(n^2 \cdot m)$. In the special case where the graph has uniform weights, the time complexity is improved to $\tilde{O}(n \cdot m \cdot d)$, where d is the maximum node degree of G.

1.2 Our approximation algorithms

The input is a hypergraph $G_h = (V_h, E_h)$ with n nodes, each $v_h \in V_h$ has weight $l(v_h) \in Z$, and m hyperedges, each $e_h \in E_h$ has weight $w(e_h) \in Z$. Let r be the maximum hyperedge size, i.e., the maximum number of nodes on a hyperedge $e \in E_h$, and d be the maximum node degree, i.e., the maximum number of hyperedges incident to a node $v \in V_h$.

We first give some definitions. The flux α of a hypergraph G_h is defined in a way similar to the way that the flux is defined on graphs. The goal is again to minimize the ratio, among all bipartitions $(\mathcal{A}, \mathcal{B})$ of V_h: sum of the weights on the hyperedges with endpoints in sets $\mathcal{A}$ and $\mathcal{B}$ over $\min\{l(\mathcal{A}), l(\mathcal{B})\}$. For simplicity, we refer to (2), for a formal definition of α on G_h. Thus, a hypergraph $G_h = (V_h, E_h)$ has flux α if every set $\mathcal{A}$ of nodes such that $l(\mathcal{A}) \leq l(V_h)/2$ is connected to the remainder of G_h with hyperedges of total weight at least $\alpha \cdot l(\mathcal{A})$. We can similarly extend the definition of the limited k-quotient separator s_k on a hypergraph G_h, and for simplicity we refer to (1) for a formal definition of s_k on G_h.

The first algorithms of this paper generalize the techniques in [21, 20] to obtain approximation algorithms for balanced bipartitioning on hypergraphs. We use *multicommodity flow arguments* to approximate, within a $O(\log(n \cdot m))$ factor, the minimum k-limited quotient hyperedge separator s_k of G_h, in $\tilde{O}((n+m)^2 \cdot m \cdot r)$ expected time. Furthermore, if the weights on the nodes of G_h are uniform, the time complexity of the approximation algorithm can be improved to $\tilde{O}((n+m) \cdot m \cdot r \cdot \max\{d, r\})$.

Theorem 1.1 still holds for the more general case when the input is a hypergraph. Then by Theorems 1.1 and 1.2 we have a polynomial time approximation algorithm for the k-balanced bipartitioning problem on hypergraphs, for any fixed $k \geq 3$. Let k and k' be fixed constants defined as in Theorem 1.1, $P(k)$ be the optimal solutions for the k-balanced bipartitioning problem on hypergraph G_h, and $l(V) = \sum_{v \in V} l(v)$.

Theorem 1.3 *The cost of our polynomial time algorithms for the k- balanced bipartitioning problem is $O(\log(n \cdot m) \cdot P(k'))$ and $O(\log(n \cdot m) \cdot \log(l(V)/l_{min}) \cdot P(k))$, respectively.*

[3]It is NP-hard to compute the flux in general graphs [27].

The time complexity of both algorithms is $O(n)$ times more than the complexity of the algorithm that approximates the flux α and the k-limited quotient hyperedge separator s_k of G_h.

In addition, we present polynomial time approximation algorithms for balanced multiway partition problems on hypergraphs.[4] We consider the multiway partitioning problem into sets of size at most k, and the problem of partitioning into sets of size at most k and at least $k/3$, for an input integer k. We present the first approximation algorithms for this problems on graphs and hypergraphs. Let P_k denote the optimal partition to the problem of partitioning G_h into sets of size at most k.

Theorem 1.4 *The cost of our polynomial time algorithm for the hypergraph partitioning problem into sets of size at most k is $O(\log(n \cdot m) \cdot \log n \cdot P_{(k+1)/4})$. If $k \leq \log n$, the cost is $O(\log(n \cdot m) \cdot \log n \cdot P_k)$.*

A similar result is also obtained for the problem of partitioning into sets of size at most k and at least $k/3$. The proposed multiway partitioning algorithms use the algorithm for the k-balanced bipartitioning problem recursively. Their time complexity is $O(\min\{\log(l(V)/k, n\} \cdot T)$, where T is the time complexity of the algorithm for the k-balanced bipartitioning problem as presented in Theorem 1.3.

We also present alternative approximation algorithms for these multiway partitioning problems that have reduced time complexity. The latter algorithms achieve polylogarithmic times optimal approximations if either k is polylogarithmic or if r is polylogarithmic. Thus the algorithms apply directly to the problem of partitioning into sets with small sizes in [15, 30]. Their time complexity is $\tilde{O}(n \cdot d \cdot m)$ if G_h has unit weights on the nodes and $\tilde{O}(n^2 \cdot m)$ in the more general case. Their approximation bound is given in the following theorem.

Theorem 1.5 *The cost of our second algorithm for the partitioning problem into sets of size at most k (and at least $k/3$) is $O(\min\{k, r\} \cdot \log^2 n \cdot P_{(k+1)/4})$. Furthermore, if $k \leq \log n$ the cost becomes $O(\min\{k, r\} \cdot \log^2 n \cdot P_k)$.*

In order to obtain Theorems 1.3, 1.4, and 1.5, we must initially transform the input hypergraph to a directed graph. Given a hypergraph $G_h = (V_h, E_h)$, we define its *directed graph representation* to be a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as follows: The set of nodes $\mathcal{V} = V \cup A$ is defined as follows: Every node $u \in V$ corresponds to a node in $u_h \in V_h$. For every hyperedge $v_h \in E_h$ there exist two vertices v^{in} and $v^{out} \in A$. Thus the nodes in set A are either labeled as *in* or as *out*. Clearly, the number of *in* nodes is equal to the number of *out* nodes and we can consider them as tuples, where tuple (v^{in}, v^{out}) corresponds to hyperedge $v_h \in E_h$. Set $\mathcal{E}$ is defined as follows: There exists a directed edge (v^{in}, v^{out}) from node v^{in} to node v^{out} for each tuple of nodes (v^{in}, v^{out}) in set A. If hyperedge $v_h \in E_h$ is incident to node $u_h \in V_h$, there exists a directed edge (u_h, v^{in}) from node $u_h \in V$ to node $v^{in} \in A$, and a directed edge (v^{out}, u_h) from node $v^{out} \in A$ to node $u_h \in V$. Therefore graph $\mathcal{G}$ has $n + 2 \cdot m$ nodes and $O(m \cdot r)$ edges. Furthermore, $w(u_h, v^{in}) = w(v^{out}, u_h) = w(v^{in}, v^{out}) = w(v_h)$.

In addition, we study the two multiway partitioning problems on planar hypergraphs. We define a hypergraph G_h to be planar if its directed graph representation $\mathcal{G}$ is planar. Let d_h be the maximum node degree of G_h, P_k be the optimal partition, and w_h be the maximum weight on a hyperedge $e \in E_h$, respectively.

Theorem 1.6 *The cost of the proposed algorithm for the partitioning problem into sets of size at most $k = O(\log n)$ (and at least $k/3$) on planar hypergraphs is $O((1 + k \cdot \sqrt{d_h \cdot w_h}/\sqrt{\log n}) \cdot P_k)$; If G_h has uniform weights on the hyperedges and $d_h = O(1)$, the cost is $O((1 + k/\sqrt{\log n}) \cdot P_k)$.*

[4] Clearly, these approximation algorithms also apply to multiway graph partitioning problems.

Therefore our planar partitioning algorithm for the above values of k, outperforms the previous approaches if the hypergraph has bounded node-degree, and uniform hyperedge weights. These restrictions apply often in VLSI.

1.3 Multicommodity and concurrent flow problems

Most of the results obtained in this paper use multicommodity flow techniques. In this section, we define the *multicommodity flow problem* and its optimization version which is called the *concurrent flow problem.* We also survey existing polynomial time algorithms for solving or approximating concurrent flow problems. This is important since the time complexity of most of our approximation algorithms is determined by the solution or approximation of a concurrent flow problem that we define later in the section.

A multicommodity flow problem MF, defined on an undirected or directed graph $G = (V, E)$ with a capacity $c(e) \in Z^+$ on every edge $e \in E$, consists of a set of *commodities.* Each commodity i is defined as a triple (s_i, t_i, d_i), where $s_i \in V$ is called the *source* of commodity i, $t_i \in V$ is called the *target* or *destination* of the commodity, and $d_i \in Z^+$ is called the *demand* of the commodity.[5] In MF the goal is to simultaneously ship the commodities from their respective *sources* to their *sinks* so that all the demands are satisfied subject to the capacity constraints on the edges of graph G. Thus a multicommodity flow problem MF can either be characterized as *feasible* or *infeasible.*

In the optimization version of MF, we call it the *concurrent flow problem* CF, the goal is to route the commodities so that we maximize the minimum, among the commodities, demand percentage routed. Given a routing $\mathcal{R}$ of the commodities, the *throughput* z is the minimum routed demand percentage. The goal in $\mathcal{CF}$ is to obtain a routing that maximizes the thoughput. Let z^* be the latter quantity, i.e., $z^* = \max_{\mathcal{R}} z$. It has been observed [20, 27] that the problem of maximizing z is equivalent to the problem of determining the minimum ratio by which the capacities must be increased in order to ship 100% of all of the demands. The latter ratio, for a given routing of the commodities, is called the *capacity utilization* λ . We use λ^* to denote the minimum capacity utilization.

Recently, Leighton, Makedon, Plotkin, Stein, Tardos and Tragoudas [20] presented the first combinatorial approximation algorithm for the concurrent flow problem for graphs with arbitrary edge capacities. Their algorithm finds a feasible flow which ships at least $(1 - \epsilon) \cdot z^*$ percent of each demand, where ϵ is a constant $0 < \epsilon < 1$. They maximize z by minimizing the capacity utilization λ. Let n, m, K, c_{max} denote the number of nodes, number of edges, number of commodities, largest demand and largest capacity on an edge of G, respectively. For a constant $\epsilon > 0$, the algorithm in [20] achieves a (1- ϵ) approximation to the concurrent flow problem in $O(n \cdot m \cdot K \cdot \log K \cdot \log^3 n)$ expected time. A deterministic version requires $O(n \cdot m \cdot K^2 \cdot \log K \cdot \log^3 n)$ time. The running time can be improved when K is large. Let K^* denote the number of different sources. In both the randomized and the deterministic algorithm we can replace K in the running time by K^* at the expense of having to replace one of the log$n$ terms by $\log(n \cdot c_{max})$. Clearly $K^* \leq n$.

In a concurrent flow problem we do not have max-flow min-cut theorems. Let (A, B) be a partition of V into two sets A and B. Let the *minimum cut ratio* S be defined as

$$S = \min_{(A,B)} \{\frac{c(A,B)}{d(A,B)}\},$$

[5] In fact, there exist generalized versions where a commodity may have more than a source or target, real demands and capacities [20]. However, in this paper we do not consider general multicommodity flows and we focus on multicommodity flows that have integer capacities and demands.

where $c(A,B)$ denotes the sum of the capacities on the edges with endpoints in different sets and $d(A,B)$ denotes the sum of the demands on the commodities for which the source and the destination are on different sets. [6] The minimum cut ratio is the analog of the minimum cut in a single commodity flow problem. [21, 20] have shown that for the all-pairs uniform demand multicommodity flow problem we have that $S/O(\log n) \leq z^* \leq S$. In addition, combining the work in [13, 20, 29, 9, 24], for any multicommodity flow problem the minimum cut ratio S and the throughput z it holds that $S/O(\log^2 K) \leq z^* \leq S$. The above results are often refered to as *approximate max-flow min-cut theorems.* The techniques used to obtain approximate max-flow min-cut theorems also guarantee approximations for the minimum cut ratio S [13, 21, 29, 9, 24].

In [21] it has been shown that an $O(\log n)$ approximation for the minimum cut ratio in the all-pairs uniform-demand concurrent flow problem leads to an approximation of the flux of a graph with uniform node-weights. Combining the above with Rao's approach in [25, 26], Leighton and Rao showed that the approximate max-flow min-cut theorems can be used to obtain approximations for the balanced bipartitioning problem on graphs [21].

1.4 The All–Pairs Concurrent Flow (APCF) problem

In this paper, we present an approximate max-flow min-cut theorem for a concurrent flow problem on G_h, we refer to it as the *All–Pairs Concurrent Flow (APCF)* problem, which allows us to obtain approximations to the hypergraph balanced bipartition problem. A concurrent flow problem, can be defined on a hypergraph by considering that a hyperedge $e \in E_h$, with capacity $c(e)$, connected nodes $v_1, ..., v_p$, behaves as a node v with capacity $c(e)$, connected to nodes $v_1, ..., v_p$ with edges of capacity at least $c(e)$. Furthermore, the minimum cut ratio S is now defined as

$$S = \min_{(A,B)} \{\frac{c(A,B)}{d(A,B)}\},$$

where the capacity $c(A,B)$ equals to the capacity on the hyperedges with endpoints in both sets A and B, and $d(A,B)$ is equal to the demand across the cut defined by the bipartition (A,B).

We now define a multicommodity flow problem on G_h that will be useful in our flux approximation algorithm. The *all-pairs concurrent flow* APCF problem on G_h consists of a commodity between every pair of nodes u_h and $v_h \in V_h$, where the source is node u_h, the destination is node v_h, and the demand is equal to $l(u_h) \cdot l(v_h)/2$. From the discussion in the previous paragraph, the capacity constraints on the hyperedges are viewed as capacity constraints on nodes. The APCF problem can also be defined on the directed graph representation $\mathcal{G}$, using arguments as in [17]: There exists a commodity for every (u,v) source-destination pair, $u, v \in V$, with demand $l(u_h) \cdot l(v_h)/2$. The capacity constraints on the edges $e \in \mathcal{E}$ are as follows. Every edge (v^{in}, v^{out}) has capacity $c(v^{in}, v^{out}) = w(v_h)$, where $v_h \in E_h$ is the hyperedge that corresponds to (v^{in}, v^{out}). Let $U_h \subseteq V_h$ be the set of nodes on hyperedge $v_h \in E_h$. For every $u \in V$ corresponding to $u_h \in U_h$ we have $c(u, v^{in}) = c(v^{out}, u) = c(v^{in}, v^{out})$. We have:

Theorem 1.7 *For the APCF problem G_h, or $\mathcal{G}$, $\Omega(S/\log(n \cdot m)) = z^* \leq S$.*

The above is a generalization of the result by Leighton and Rao [21] to hypergraphs. One can easily combine the techniques used to derive Theorem 1.7 and the approach in [24] to obtain an approximate

[6] It is NP-hard to compute S optimally [27].

max-flow min-cut theorem for any multicommodity flow problem on G_h with K commodities. [7] (These generalizations are not considered here since they are not related to hypergraph partitioning.)

This paper is organized as follows. In Section 2 we prove Theorem 1.7 and we show how to derive an approximation algorithm for balanced bipartitions as stated in Theorem 1.3. Our approach for obtaining Theorem 1.7 uses duality arguments for concurrent flows on hypergraphs [21]. The dual of a concurrent flow problem defined on a graph $G = (V, E)$ has been formulated in [11, 27, 21]. In Section 2.1 we derive similar duality arguments for a concurrent flow problem $\mathcal{CF}$ defined on a hypergraph G_h. In Section 2.2, we use the dual formulation of the $\mathcal{APCF}$ problem to obtain polylogarithic times optimal approximations for the balanced bipartition problem on G_h. Observe here, that in order to use the duality arguments for the $\mathcal{ACPF}$ problem, we must first derive $(1-\epsilon)$ approximations for the maximum throughput z^*, for a constant $0 < \epsilon < 1$. (We use the algorithms in [20] to approximate the $\mathcal{APCF}$ problem because they outperform the alternative methods in [31, 27, 7, 6, 14].) Finally, in Section 3 we propose the first approximations for graph and hypergraph multiway partitioning problems. As Theorems 1.4, 1.5 and 1.6 show, there is a trade-off between worst case time complexity and performance guarantee among the alternatives. Section 4 concludes.

2 An approximation for the flux (and the balanced bipartition problem)

We first formulate the dual of any concurrent flow problem on the input hypergraph $G_h = (V_h, E_h)$. Then, in Section 2.2, we use the duality arguments to obtain a polynomial time algorithm that approximates the flux α of G_h within a $(O\log(n \cdot m))$ factor. The definitions of s_k and α respectively on hypergraph G_h (as in (1) and (2), respectively), guarantee that a polylogarithmic approximation to the flux α suffices to approximate s_k within the same polylogarithmic factor. We can therefore use as a final step Rao's approach in [25, 26] to obtain polynomial time algorithms for the k- balanced bipartitioning problem with costs $O(\log(n \cdot m) \cdot P(k'))$ and $O(\log(n \cdot m) \cdot \log(l(V)/l_{min}) \cdot P(k))$, respectively.

2.1 The dual of a concurrent flow problem on a hypergraph

In this section, we show a generalization of the duality arguments on multicommodity flow problems [20, 11, 27] to the case of hypergraphs. Namely, we show that the dual equations of any concurrent flow problem $\mathcal{CF}$ on G_h that has K commodities describe the problem of assigning weights to the hyperedges of the input hypergraph G_h so that the sum over all commodities: distance from the commodity source to the commodity destination multiplied by the demand of the commodity is maximized. Then we interpret the above duality arguments for the $\mathcal{APCF}$ problem.

We obtain the dual equations for the $\mathcal{CF}$ problem by working on $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the directed graph representation of the input hypergraph $G_h = (V_h, E_h)$. For convenience, we formulate the $\mathcal{CF}$ problem as the problem of minimizing the capacity utilization λ.

We first construct the linear program of the latter minimization problem. Let $\text{inc}(u) = \bigcup_{(v,u)\in\mathcal{E}} v$, $\text{out}(u) = \bigcup_{(u,v)\in\mathcal{E}} v$. Furthermore, let $f_i(u, v)$ be the flow of commodity i on edge (u, v) from node u to node v, and d_i denote the demand of commodity i with source s_i and sink t_i. In the linear program we minimize λ, subject to

[7] In this case we have that $\Omega(S/(\log^2 n \cdot K)) = z^* \leq S$.

$$\sum_{i=1}^{K}(f_i(u^{in},u^{out})+f_i(u^{out},u^{in})) \leq \lambda \cdot c(u^{in},u^{out}),\ \forall \text{ pair of nodes } u^{in},u^{out} \in A. \quad (3)$$

and

$$if\ v \neq \{s_i \vee t_i\},\ \sum_{u \in \text{inc}(v)} f_i(v,u) - \sum_{u \in \text{out}(v)} f_i(u,v) = 0,\ \forall v \in V,\ \forall \text{ commodity } i$$

$$if\ v = s_i,\quad \sum_{u \in \text{inc}(v)} f_i(v,u) - \sum_{u \in \text{out}(v)} f_i(u,v) = -d(i),\ \forall v \in V,\ \forall \text{ commodity } i \quad (4)$$

$$if\ v = t_i,\quad \sum_{u \in \text{inc}(v)} f_i(v,u) - \sum_{u \in \text{out}(v)} f_i(u,v) = d(i),\ \forall v \in V,\ \forall \text{ commodity } i$$

The constraints in (3) are due to the capacity constraints on $\mathcal{G}$. This set of inequalities guarantees that we do not exceed the scaled by λ capacity of any edge (u^{in},u^{out}), $\forall$ pair of nodes $u^{in},u^{out} \in A$. From the construction of graph $\mathcal{G}$, we do not need to impose similar capacity constraints on the edges not listed in (3). The constraints in (4) express flow continuity conditions. These equations ensure that each node s_i is the source of d_i units for commodity i, and that each node t_i is the destination of d_i units for commodity i. This guarantees that all the flows are going from the sources to the destinations.

The dual for the linear program is as follows. We introduce dual variables $\ell(u^{in},u^{out})$ one for each directed edge (u^{in},u^{out}), $\forall u^{in},u^{out} \in A$, and ℓ_{i,s_i}, ℓ_{i,t_i} correspond to the constraints in (4).

$$\text{Maximize} \sum_{i=1}^{K}(\ell_{i,t_i}-\ell_{i,s_i}) \cdot d_i \quad (5)$$

$$\ell_{i,v}-\ell_{i,u}+\ell(u^{in},u^{out}) \geq 0,\quad \forall u^{in},u^{out} \in A \quad (6)$$

$$\sum_{\forall u^{in},u^{out} \in A} c(u^{in},u^{out}) \cdot \ell(u^{in},u^{out}) \leq 1,\quad \text{where } \ell(u^{in},u^{out}) \geq 0. \quad (7)$$

Inequality (6) implies that ℓ_{i,t_i} can not be larger than the length of the shortest path, with respect to a length $\ell(u^{in},u^{out})$ assigned to edge (u^{in},u^{out}) from node u^{in} to node u^{out}, plus the value ℓ_{i,s_i}. Furthermore, due to the maximization of the function in (5), $\ell_{i,t_i} = \ell_{i,s_i} + \text{dist}_\ell(s_i,t_i)$, where $\text{dist}_\ell(s_i,t_i)$ is the shortest path between s_i and t_i subject to the length function $\ell(s_i,t_i)$. From the latter equality, we rewrite the above set of equations as follows.

$$\text{Maximize} \sum_{i=1}^{K} \text{dist}_\ell(s_i,t_i) \cdot d_i \quad (8)$$

$$\sum_{\forall u^{in},u^{out} \in A} c(u^{in},u^{out}) \cdot \ell(u^{in},u^{out}) \leq 1,\quad \text{where } \ell(u^{in},u^{out}) \geq 0. \quad (9)$$

The maximum value of the above linear program is equal to λ^*. In general, $\lambda = 1/z$ is less than or equal to $1/(\sum_{i=1}^{K} \text{dist}_\ell(s_i,t_i) \cdot d_i)$, where $\sum_{\forall u^{in},u^{out} \in A} c(u^{in},u^{out}) \cdot \ell(u^{in},u^{out}) = 1$. Let ℓ^* be the length function that minimizes $1/(\sum_{i=1}^{K} \text{dist}_\ell(s_i,t_i) \cdot d_i)$. We have:

Theorem 2.1 *The dual of a $\mathcal{CF}$ problem with K commodities maximizes $\sum_{i=1}^{K} \text{dist}_\ell(s_i,t_i) \cdot d_i$, for a length function ℓ on edges (u^{in},u^{out}), $u^{in},u^{out} \in A$, such that $\sum_{\forall u^{in},u^{out}} c(u^{in},u^{out}) \cdot \ell(u^{in},u^{out}) = 1$. Furthermore, at optimality, there exists a length function ℓ^* so that the minimum capacity utilization λ^* is equal to $1/(\sum_{i=1}^{K} \text{dist}_{\ell^*}(s_i,t_i) \cdot d_i)$ under the constraint that $\sum_{\forall u^{in},u^{out}} c(u^{in},u^{out}) \cdot \ell^*(u^{in},u^{out}) = 1$.*

The above theorem is expressed on the directed graph representation $\mathcal{G}$. However, it can be translated in a straightforward way on the input hypergraph G_h. The dual maximizes $\sum_{i=1}^{K} \text{dist}_{\ell^*}(s_i, t_i) \cdot d_i$, for a length function ℓ on the hyperedges such that $\sum_{\forall u_h \in E_h} c(u_h) \cdot \ell^*(u_h) = 1$. We now translate Theorem 2.1 to the $\mathcal{APCF}$ problem:

Theorem 2.2 *The dual of the $\mathcal{APCF}$ problem on $\mathcal{G}$ maximizes $\sum_{\forall u,v \in V} \text{dist}_{\ell}(u,v) \cdot l(u_h)$, for a length function ℓ on the directed edges (w^{in}, w^{out}), $w^{in}, w^{out} \in A$, such that $\sum_{\forall w^{in}, w^{out}} c(w^{in}, w^{out}) \cdot \ell(w^{in}, w^{out}) = 1$. Furthermore, at optimality, there exists a length function ℓ^* so that the minimum capacity utilization λ^* is equal to $1/(\sum_{\forall u,v \in V} \text{dist}_{\ell^*}(u,v) \cdot l(u_h))$, under the constraint that $\sum_{\forall w^{in}, w^{out} \in A} c(w^{in}, w^{out}) \cdot \ell^*(w^{in}, w^{out}) = 1$.*

2.2 A polynomial time approximation algorithm for the flux

In this section, we present a polynomial time approximation algorithm for the flux α of a hypergraph $G_h = (V_h, E_h)$.

In order to approximate the flux α of the input hypergraph $G_h = (V_h, E_h)$ we show that it suffices to obtain an approximate max-flow min-cut theorem for the $\mathcal{APCF}$ problem on the directed graph representation $\mathcal{G}$. For simplicity, we use S and z^* to denote the minimum cut ratio and the maximum throughput, respectively.

Consider graph $\mathcal{G}$, and a set $\mathcal{C} \subseteq A$ such that a node v^{in} that corresponds to a node $v_h \in E_h$ is in $\mathcal{C}$ if and only if v^{out} is in $\mathcal{C}$. We say that the triple $(\mathcal{V}_1, \mathcal{V}_2, \mathcal{C})$ is a *separation* of $\mathcal{G}$ if and only if the removal of set $\mathcal{C}$ disconnects $\mathcal{V}$ into two sets $\mathcal{V}_1$ and $\mathcal{V}_2$. Clearly, for any bipartition $(\mathcal{A}, \mathcal{B})$ of V_h we have a corresponding separation $(\mathcal{V}_1, \mathcal{V}_2, \mathcal{C})$ on the directed graph $\mathcal{G}$. Each separation $(\mathcal{V}_1, \mathcal{V}_2, \mathcal{C})$ on $\mathcal{G}$ defines a cut, which we denote by $\Gamma(\mathcal{C})$, with capacity $c(\Gamma(\mathcal{C}))$ equal to the sum of the capacities on the directed edges (v^{in}, v^{out}) between all the nodes v^{in} and v^{out} in $\mathcal{C}$. From the construction of $\mathcal{G}$, the cost

$$\sum_{e_h \in E_h : e_h \text{ on } u_h \in \mathcal{A} \wedge \text{on } v_h \in \mathcal{B}} w(e_h)$$

of the bipartition $(\mathcal{A}, \mathcal{B})$ on G_h equals to the capacity $c(\Gamma(\mathcal{C}))$ related to the separation $(\mathcal{V}_1, \mathcal{V}_2, \mathcal{C})$ of $\mathcal{G}$.

Let $d(\Gamma(\mathcal{C})) = l(\mathcal{A}) \cdot l(\mathcal{B})$ be the demand across the cut $cut(\mathcal{C})$ in the $\mathcal{APCF}$ problem on $\mathcal{G}$. We also define the *cut ratio* $S_{\mathcal{C}}$ (associated to a cut $cut(\mathcal{C})$) as follows. Let $e_h \in E_h$, $u_h \in \mathcal{A}$, and $v_h \in \mathcal{B}$:

$$S_{\mathcal{C}} = c(\Gamma(\mathcal{C}))/d(\Gamma(\mathcal{C})) = \sum_{e_h : e_h \text{ on } u_h \wedge \text{on } v_h} \frac{w(e_h)}{l(\mathcal{A}) \cdot l(\mathcal{B})}.$$

Clearly, $S = \min_{\mathcal{C}} S_{\mathcal{C}}$.[8] Furthermore, for any concurrent flow problem we have that

$$z^* \leq S. \tag{10}$$

Assume that $l(\mathcal{A}) \leq l(\mathcal{B})$. Then $\alpha = \Theta(\min\{c(\Gamma(\mathcal{C}))/l(\mathcal{A})\})$. Thus, from the upper and lower bounds on the value of $l(\mathcal{B})$, we have that the flux α and the minimum cut ratio S are related as:

$$\alpha = \Theta(l(V_h) \cdot S). \tag{11}$$

We approximate the flux a within a $\log(n \cdot m)$ factor by first showing that in the $\mathcal{APCF}$ problem on graph $\mathcal{G}$ $z^* = S/\log(n \cdot m)$. This is also often referred to as an approximate max-flow min-cut theorem

[8] It is NP-hard to compute the minimum cut ratio S [27].

for the $\mathcal{APCF}$ problem on $\mathcal{G}$. (See also [21, 20].) Then we use arguments similar to the ones in [21], and we modify our approach that obtains the lower bound so that we can find a partition with cost within an $O(\log(n \cdot m))$ factor to the flux α of G_h.

We show that $z^* = \Omega(S/(\log(n \cdot m))$, using duality arguments for the $\mathcal{APCF}$ problem where we consider lengths on the edges of the directed graph $\mathcal{G}$. (See Theorem 2.2.) Consider a nonnegative function ℓ^* on the edges of $\mathcal{G}$ that maximizes (approximately) the throughput z on $\mathcal{CF}$. (Such a function can be derived using the techniques in [20].) Let $\text{dist}_{\ell^*}(u, v)$ be the shortest path distance between nodes $u \in V$ and $v \in V$ under the length function ℓ^* on graph $\mathcal{G}$, $\text{length}(P(s_i, t_i))$ be a function denoting the length of path $P(s_i, t_i)$ from node $s_i \in V$ to node $t_i \in V$ subject to the length function ℓ^* on the directed graph $\mathcal{G}$. Finally, let $l(s_{ih})$ and $l(t_{ih})$ denote the weights on the nodes s_{ih} and t_{ih} of G_h that correspond to nodes s_i and t_i in the directed graph representation G_h.

We present an algorithm, we refer to it as $ROUTE$, that routes the commodities so that

$$\sum_i \text{length}(P(s_i, t_i)) \cdot d(s_{ih}, t_{ih}) = \sum_i \text{length}(P(s_i, t_i)) \cdot \frac{l(s_{ih}) \cdot l(t_{ih})}{2} =$$

$$= O(\frac{\log(n \cdot m)}{S}) = O(\frac{l(V_h) \cdot \log(n \cdot m)}{\alpha}). \tag{12}$$

This establishes an approximate max-flow min-cut theorem for the $\mathcal{ACPF}$ flow problem on $\mathcal{G}$ (and thus on G_h). Combining (10), (11), (12), and Theorem 2.2, we deduce that

$$\Omega(S/\log(n \cdot m)) = z^* \leq S.$$

Furthermore, in order to approximate the flux α, we only need to modify algorithm $ROUTE$ as Leighton and Rao did in [21]. The algorithm keeps track of α_{min}, the minimum hyperedge expansion observed while it runs. We show that (12) holds even if we substitute α by α_{min}. Using similar arguments as above (for the derivation of the lower bound for z^*) we can easily deduce that α_{min} is $O(\log(n \cdot m)) \cdot \alpha$, i.e., it is a logarithmic approximation for the flux α of the hypergraph G_h.

We observe that our approach obtains a logarithmic times optimal approximation of the flux α. Therefore we can afford obtaining a constant times optimal approximation for the $\mathcal{APCF}$ flow problem. This can be obtained by the algorithm in [20] which returns a function ℓ on the edges of $\mathcal{APCF}$. Although the algorithm in [20] does not satisfy the constraint $\sum_{e \in \mathcal{E}} c(e)\ell(e) = 1$, we can scale down the lengths to satisfy the latter constraint. For the sake of simplicity, we use ℓ to denote the scaled length function as well.

We must also reduce the capacities of the hyperedges appropriately. Let α_v be the hyperedge expansion of a single node v, i.e., the ratio sum of the capacities on the incident hyperedges over $l(v)$. We can assume that there is no capacity $c(v^{in}, v^{out})$ on an edge (v^{in}, v^{out}) of the directed graph $\mathcal{G}$ with cost more than $2 \cdot l(V_h) \cdot \alpha^m$, where $\alpha^m = \min\{\alpha_v\}$. Such an edge will never participate in the computation of the flux α. If a node had cost more than $2 \cdot l(V_h) \cdot \alpha^m$, we could have reduced its cost to the latter quantity without affecting the computation of the flux α. Let $C(E_h)$ be the sum of the (reduced) capacities on the hyperedges of G_h. Next, we describe $ROUTE$.

Algorithm ROUTE

Step 1: Split each edge (v^{in}, v^{out}) into discrete pieces by placing $\max\{1 + \lceil \ell(v^{in}, v^{out}) \cdot C(E_h) \rceil\}$ tokens, each having capacity $c(v^{in}, v^{out})$. Each piece has size $C(E_h)^{-1}$. Furthermore, the sum of the capacities on the tokens is $C^{tok} = \sum_{(v^{in}, v^{out})} c(v^{in}, v^{out}) \cdot (1 + \ell(v^{in}, v^{out})C(E_h)) \leq 2 \cdot C(E_h)$. (The latter inequality is due to the constraint $\sum_e c(e) \cdot \ell(e) = 1$ in the dual formulation of $\mathcal{APCF}$ problem.)

Step 2: (a) Select a node $v \in V$ such that the weight $l(v_h)$ of the respective node $v_h \in V_h$ is maximum, to initiate a set $\mathcal{S}$ that contains nodes and tokens; Enhance set $\mathcal{S}$ by incorporating all the tokens incident (in breadth first search expansion manner) to node v^{in}; Initialize $i := 1$.

(b) Extend set $\mathcal{S}$ by a level of tokens, expanding in a breadth first search manner along the directed edges of $\mathcal{G}$; Let i, C_i be the depth of the set measured in number of tokens, and the total capacity on the tokens in the set at level i, respectively; If $C_i > (1 + \alpha \cdot \mathcal{D}/\ 4 \cdot l(V_h) \cdot C(E_h))$ then go to Step 2(b).

(c) Let $V_{\mathcal{S}}$ be the set of nodes of V which are also in $\mathcal{S}$, and $V_{\mathcal{S}h}$ be the set of nodes in V_h which correspond to the set of nodes $V_{\mathcal{S}}$. Set $V_{\mathcal{S}}$ defines a bipartition of V_h. If $l(V_{\mathcal{S}h}) \geq l(V_h)/2$ then go to Step 3 else discard the nodes in set $V_{\mathcal{S}}$ and go to Step 1.

Step 3: Expand set $\mathcal{S}$ in a breadth first search manner, a level of tokens at a time to reach all the remaining nodes in V. Now we have a directed breadth first search tree which determines a unique path for each pair of commodities (v^{in}, u^{out}).

End of ROUTE

We show that if the route each commodity along the indicated path we guarantee the upper bound of (12). First we prove that the condition at Step 2(c) will succeed at some iteration of algorithm *ROUTE*.

Lemma 2.1 *Algorithm ROUTE always terminates.*

Proof: By contradiction, assuming that the condition at Step 2(c) never succeeds. Assume that the union of the removed sets at Step 2(c) is at least $l(V_h)/4$. Since none of these sets has size $l(V_h)/2$, their union is in the range $[l(V_h)/4, 3 \cdot l(V_h)/4]$. Let $c(\mathcal{S})$ denote the sum of the capacities on the tokens where the expansion failed on the condition at Step 2(c). By the definition of the flux α, we have that

$$\sum_{\mathcal{S}} c(\mathcal{S}) \geq \frac{\alpha \cdot l(V_h)}{4}. \tag{13}$$

However, the expansion condition at Step 2(b) guarantees that $c(\mathcal{S}) \leq \alpha \cdot \mathcal{D} \cdot C_{\mathcal{S}}^{tok}/(4 \cdot l(V_h) \cdot C(E_h)$, where $C_{\mathcal{S}}^{tok}$ denotes the sum of the capacities on the tokens in $\mathcal{S}$. From the latter we deduce that $\sum_{\mathcal{S}} c(\mathcal{S}) \leq (\alpha \cdot \mathcal{D}/(4 \cdot l(V_h) \cdot C(E_h)) \cdot \sum_{\mathcal{S}} C_{\mathcal{S}}^{tok}$. The term $\sum_{\mathcal{S}} C_{\mathcal{S}}^{tok}$ cannot exceed $2 \cdot C(E_h)$. Therefore we have that $\sum_{\mathcal{S}} c(\mathcal{S}) \leq \alpha \cdot \mathcal{D}/(2 \cdot l(V_h))$, and since $\mathcal{D} \leq l(V_h)^2/2$, we have that $\sum_{\mathcal{S}} c(\mathcal{S}) < \alpha \cdot l(V_h)/4$, a contradiction to (13). □

In order to prove the upper bound on (12), we need to show four lemmas. The first lemma refers to the expansion Step 2(b) of *ROUTE*:

Lemma 2.2 $\epsilon = \alpha \cdot \mathcal{D}/(4 \cdot l(V_h) \cdot C(E_h)) < 1$.

Proof: Considering node v, we have that $\alpha \leq \alpha_v$. Therefore

$$\alpha \cdot l(V_h) \leq 2 \cdot C(E_h). \tag{14}$$

Since $\mathcal{D} < l(V_h)^2/2$ we have that $\epsilon < \alpha \cdot l(V_h)/(8 \cdot C(E_h))$. The latter combined with (14) show the lemma. □

Let $C_{\mathcal{S}}^{in}$ denote the sum of the capacities of the tokens at the first level of expansion, i.e., the sum of the capacities of the tokens incident to node v that initiates a set $\mathcal{S}$ which satisfies the expansion condition at Step 2(c). The next lemma shows that $C_{\mathcal{S}}^{in}$ is a large portion of $C(E_h)$.

Lemma 2.3 *We have that* $C(E_h) = O(n \cdot m) \cdot C_{\mathcal{S}}^{in}$.

Proof: After the capacity reductions on the hyperedges, $C(E_h) = O(m \cdot l(V_h) \cdot \alpha^m)$. Therefore the node v that initiates sets $\mathcal{S}$ satisfies that $C(E_h) = O(m \cdot l(V_h) \cdot a_v) = O(m \cdot l(V_h) \cdot C^{in}_{\mathcal{S}}/l(v))$. Let $l(V')$, n' be the total node weight, and the number of nodes of the induced hypergraph when algorithm *ROUTE* selected node v to generate set $\mathcal{S}$. The selection of node v guarantees that $l(v) \geq l(V')/n' \geq l(V_h)/(2 \cdot n)$. The lemma follows from the above inequalities. □

Lemma 2.4 *The depth of the breadth first search tree at the end of Step 2 is* $O(l(V_h) \cdot \log(n \cdot m)/(\alpha \cdot \mathcal{D}))$.

Proof: By a contradiction argument. Assume that the depth d of the generated tree does not satisfy the lemma. Then from the expansion condition at Step 2(c) we have that $C_d > (1 + (\alpha \cdot \mathcal{D}/(4 \cdot l(V_h) \cdot C(E_h))))^{d-1} \cdot C^{in}_{\mathcal{S}}$, where C_d is the sum of the capacities on the tokens in set $\mathcal{S}$ at breadth first search depth d. By Lemma 2.3, it suffices to show that $(1 + (\alpha \cdot \mathcal{D}/\ (4 \cdot l(V_h) \cdot C(E_h))))^{d-1} = \Omega(nm)$. This can be easily proven as in [13, 21], by taking the logarithms of both parts of the latter inequality, and by using Lemma 2.2. □

From Lemma 2.4 we have that, for all commodities (s_{ih}, t_{ih}) routed during Step 2, $\sum_i \text{length}(P(s_i, t_i))$ $d(s_{ih}, t_{ih}) = O(l(V_h) \cdot \log(n \cdot m)/\alpha)$. The following lemma shows that for all commodities (s_{jh}, t_j) routed during Step 3, we have that $\sum_i \text{length}(P(s_j, t_j)) \cdot d(s_{jh}, t_{jh}) = O(l(V_h) \cdot \log(n \cdot m)/\alpha)$.

Lemma 2.5 *For all commodities* (s_{ih}, t_{ih}) *routed during Step 3 we have that* $\sum_i \text{length}(P(s_i, t_i)) \cdot d(s_{ih}, t_{ih})$ $= O(l(V_h) \cdot \log(n \cdot m)/\alpha)$.

Proof:

For each pair of nodes u and v, let $P(u, v)$ be the unique path between u and v through the breadth-first search tree. Let $tlength(P(u, v))$ denote the token distance along this path. We show that

$$\sum_i length(P(u, v))l(u)l(v) is O(C \cdot \log(n \cdot m)/\alpha). \tag{15}$$

This suffices to show the lemma.

Let $L(u)$ denote the number of breadth-first search levels of Step 2 before u is reached. (If u belongs to T, then $L(u) = 0$.) Since T has token depth at most $O(C \log(n \cdot m)/\alpha$, we have that

$$tlength(P(u, v)) = O(l(V_h) \cdot \log(n \cdot m) \cdot C/\alpha) + L(u) + L(v).$$

Hence

$$\sum_{u,v} tlength(P(u, v)) \cdot d(s_{ih}, t_{ih}) = O(tlength(P(u, v)) \cdot l(u) \cdot l(v)) =$$

$$= O(\sum_{u,v} [l(V_h) \cdot C \log(n \cdot m)/\alpha + L(u) + L(v)] \cdot l(u) \cdot l(v)) =$$

$$= O(l(V_h) \cdot \log(n \cdot m) \cdot C/\alpha) + O(\sum_{u,v} [L(u) + L(v)] \cdot l(u) \cdot l(v)).$$

Thus, we have that

$$\sum_{u,v} tlength(P(u, v)) \cdot d(s_{ih}, t_{ih}) = O(l((V_h) \cdot \log(n \cdot m) \cdot C/\alpha) + O(l(V) \cdot \sum_u (L(u) \cdot l(u))). \tag{16}$$

In the final step of our proof, we observe that if at level i there is still $l(u)$ weight on nodes that have not been yet reached then the weight of the tokens at level i must be at least $\alpha \cdot L(u)$. (The latter is due to the definition of the flux α.) Since the total the total token weight is at most $2 \cdot C$, we have that

$$\sum_u L(u) \cdot l(u) \leq 2 \cdot C/\alpha. \tag{17}$$

Observe that (17) together with (16), give (15), and thus the proof of our lemma. □

We observe when substituting α by α_{min} (the minimum observed hyperedge expansion while *ROUTE* runs) in the expansion condition of Step 2(c), we obtain an equation similar to (12). The only difference is that now α is substituted by α_{min}. The latter combined with (11), Theorem 2.2, and the fact that $z^* \leq S$, results to the following theorem.

Theorem 2.3 *The minimum observed hyperedge expansion α_{min}, while the modified* ROUTE *executes, has cost no more than $O(\alpha \cdot \log(n \cdot m))$.*

3 Multiway hypergraph partitioning

3.1 An algorithm for the general case

In this section we consider the problem of partitioning the input hypergraph $G_h = (V_h, E_h)$ into sets of size at most k, for some input integer $k \leq l(V_h)/3$. We present the first polylogarithmic times optimal, polynomial time, approximation algorithm for this partitioning problem. The algorithm can also provide approximations to the problem of partitioning G_h into sets of size at most k and at least $k/3$. In the following, we describe the algorithm for the problem of partitioning into sets of size at most k, we prove its performance guarantee and we show how it also applies to the case when we impose a lower bound of $k/3$ for each set in the partition. Observe here that, in order for the problem to be feasible, every node must have weight at most k. Therefore $l(V) \leq n \cdot k$.

First, we present and prove two lemmas. Consider hypergraph $G_{hi} = (V_{hi}, E_{hi})$, induced by the set of nodes $V_{hi} \subseteq V_h$. Let ${P_k}^i$ and ${P_3}^i$ denote the optimal partition into sets of size at most k and the optimal 3-balanced bipartition of G_{hi}, respectively.

Lemma 3.1 *If $k \leq l(V_{hi})/3$, then ${P_3}^i \leq {P_k}^i$.*

Proof: Assume that the optimal partition of the nodes V_{hi} of a hypergraph G_{hi} into sets of size at most k consists of p subsets a_i, $1 \leq i \leq p$. In order to prove the lemma, it suffices to show that we can assign the p subsets into two sets, named V_{hi1} and V_{hi2}, so that both $l(V_{hi1})$ and $l(V_{hi2})$ are in the range $[l(V_{hi})/3, (2 \cdot l(V_{hi})/3]$ and without breaking any set. To see this, we first sort the p subsets in increasing order, according to their sizes. Then, while scanning the sorted list, we alternatively assign each subset a_i, $1 \leq i \leq p$, into the two sets V_{hi1} and V_{hi2}, starting assigning to the set V_{hi1}. Clearly, $l(V_{hi1}) \geq l(V_{hi2})$. This is because we have that $l(a_1) \geq l(a_2)$, $l(a_3) \geq l(a_4)$, ..., and thus we have that $l(V_{hi1}) \geq l(V_{hi})/3$. To prove the lemma, it suffices to show that $l(V_{hi1}) \leq 2l_i/3$. Since $l(a_2) \geq l(a_3)$, $l(a_4) \geq l(a_5)$, ..., it follows that $l(V_{hi1})$ - $l(a_1) \leq l(V_{hi2})$. The lemma follows since $l(V_{hi2}) = l(V_{hi})$ - $l(V_{hi1})$, and $l(a_1) \leq l(V_{hi})/3$. □

From the proof of Lemma 3.1 we have:

Corollary 3.1 *Given a partition P_{hk} of a hypergraph $G_h = (V_h, E_h)$ into sets of size at most $k \leq V_h/3$, we can always form a 3-balanced bipartition of G_h, without splitting any set in P_{hk}.*

Suppose now that the input hypergraph $G_h = (V_h, E_h)$ has been partitioned (by some method) into p subhypergraphs $G_{hi} = (V_{hi}, E_{hi})$, $1 \leq i \leq p$. Let $P_3{}^i$ be the optimal 3-balanced bipartition of subhypergraph G_{hi}, and P_k be the cost of the optimal partition of G_h into sets of size at most k. For simplicity, let P_k also denote the edges that form the optimal partition into sets of size at most k. We have the following lemma:

Lemma 3.2 *If* $k \leq \min_i\{l(V_{hi})\}$ *then* $P_k \geq \sum_{i=1}^{p} P_3{}^i$.

Proof: We show that we can partition each one of the p subhypergraphs G_{hi} into sets of size in the range $[l(V_{hi})/3, 2 \cdot l(V_{hi})/3]$, by removing only a subset of the edges in the optimal partition P_k. For each each set V_{hi}, we remove the edges in $\{P_k \cap E_{hi}\}$. The removal of these edges partitions the subhypergraph G_{hi} into sets of size at most $k \leq l(V_{hi})/3$. Thus, Lemma 3.1 applies for G_{hi}. From Corollary 3.1, we know that we have also obtained a 3-balanced bipartition of G_{hi}. The same argument works for all the subhypergraphs G_{hi} simultaneously. □

We are now ready to present our multiway partitioning algorithm, we call it *MULTIWAY*, into sets of size at most k. Let n be the number of nodes of the input hypergraph G_h.

Algorithm MULTIWAY

Step 1 If $k > \log n$, apply the balanced hypergraph bipartitioning algorithm of the previous section recursively, each time doing a 3-balanced bipartition on any subhypergraph with size more than k. If all sets have size no more than k exit.

Step 2 If $k \leq \log n$, apply the 3-balanced hypergraph bipartitioning algorithm for each set of size more than $3 \cdot \log n$. Now each resulting set a_i has size greater than or equal to $\log n$. Use an exponential algorithm to partition each set a_i into sets of size at most k.

End of algorithm MULTIWAY

Observe that the cost of each balanced bipartition of any induced hypergraph while we are iterating at either Step 1 or 2 is $O(\log(n \cdot m)$ times the optimal 3-balanced bipartitioning for that subhypergraph, where n, m are the number of nodes and the number hyperedges of G_h, respectively. Let APP_k be the partition (and its cost) that *MULTIWAY* obtains for the problem of partitioning into sets of size at most k. We have:

Theorem 3.1 *If* $k > \log n$ *then* $APP_k = O(\log(n \cdot m) \cdot \log n \cdot P_{(k+1)/4})$; *Otherwise,* $APP_k = O(\log(n \cdot m) \cdot \log n \cdot P_k)$.

Proof: Consider all the internal nodes at level i, $1 \leq i \leq O(\log(l(V_h)/k) = \log n$, of the partition tree formed while algorithm *MULTIWAY* runs. These nodes form a collection of sets V_i for which Lemma 3.2 applies when $k \leq l(V_i)/3$. The balanced bipartitioning algorithm of the previous section only guarantees a 3-balanced bipartitioning with cost $O(\log(n \cdot m))$ from the optimal 3-balanced bipartition of any hypergraph G_{hi}. Furthermore, since we want to ensure that no set has size more than k, we perform a 3-balanced bipartition even for a set of size $k + 1$. In this case, we may end up into sets of size $(k + 1)/3$. Thus, from the performance guarantee of the balanced bipartitioning algorithm, we can only compare with $P_{(k+1)/4}$, the optimal cost of partitioning into sets of size at most $(k + 1)/4$, and the cost of *MULTIWAY* is $O(\log(n \cdot m) \cdot \log n) \cdot P_{(k+1)/4})$.

In the case where $k = O(\log n)$, we apply 3-balanced bipartitions on any subhypergraph with $O(\log n)$ nodes. Let G_{hi} be any such internal node of the partition tree. Let P_k be the optimal partition of the

input hypergraph G_h into sets of size at most k, and P_k^i be the optimal partition of G_{hi} into sets of size at most k. Clearly, $\bigcup_i P_k^i$, the union of optimal partitions of hypergraphs G_{hi} into sets of size at most k, partitions the input hypergraph G_h into sets of size at most k. We have that $\bigcup_i P_{ki} \leq P_k$. Furthermore, Lemma 3.2 holds for all the $O(\log(l(V_h)/k)) = O(\log n)$ levels in the partition tree while we were applying balanced bipartitioning. (The argument is the same as in the previous case.) Combining the above we have that if $k = O(\log n)$, then the cost of *MULTIWAY* is $O(\log(n \cdot m) \cdot \log n) \cdot P_k)$. □

Algorithm *MULTIWAY* also applies to the case when each set must satisfy a lower bound of $k/3$, in addition to the upper bound k. Both Lemmas 3.1 and 3.2 hold in this case. Furthermore, the 3-balanced bipartitioning algorithm always preserves the lower bound. The time complexity of MULTIWAY is $O(\log n \cdot T)$, where T is the complexity of the balanced bipartition algorithm presented in the previous section.

3.2 An alternative approach

In this section we show that the algorithm of the previous section can be modified to derive approximations for special cases of multiway hypergraph partitioning. The bounds derived will be nontrivial if either k (the maximum set size), or r (the maximum hyperedge size) is $o(n)$. We will call this algorithm $MULTIWAY1$. Let $G_h = (V_h, E_h)$ be a hypergraph with n nodes and m hyperedges.

Algorithm MULTIWAY1

Step 1 Given $G_h = (V_h, E_h)$, remove all the hyperedges that connect more than k nodes. Let D_1 be the set of the removed hyperedges. Let $G_{h1} = (V_h, E_{h1})$ be the resulting hypergraph.

Step 2 Represent G_{h1} by a graph G', in which every hyperedge that connects r nodes is substituted by a chain of length r - 1 that connects these r nodes. In G', merge all possible multiple edges in one weighted edge.

Step 3 Apply the generalization of Leighton-Rao's algorithm as presented in [20] on G' recursively, exactly as we applied the algorithm of the previous section on the input hypergraph, to end up with sets of size at most k. Compute the number of hyperedges that connect nodes in different sets. Let D_2 be the set of these hyperedges. The hyperedges in D_1 together with the hyperedges in D_2 form the solution to our partitioning problem.

End of MULTIWAY1

Let APP_k be the partition (and its cost) obtained by algorithm $MULTIWAY1$, and P_k be the optimal partition (and its cost) for partitioning G_h into sets of size at most k. Similarly, let APP'_k be the partition (and its cost) obtained by the algorithm in [20, 29] for partitioning G' into sets of size less than or equal to k, and P'_k be the optimal partition (and its cost). Finally, we use r to denote the number of nodes in the largest hyperedge of G_h. The solutions obtained by algorithm $MULTIWAY1$ are bounded by the following theorem.

Theorem 3.2 $APP_k \leq O(\min\{k, r\} \cdot \log^2 n) \cdot P_{(k+1)/4})$. *If* $k \leq \log n$, *then* $APP_k \leq O(\min\{k, r\} \cdot \log^2 n) \cdot P_k)$.

Proof: Clearly, all the removed hyperedges at Step 1 are participating in any partition and thus in the optimal. We now prove the theorem for the case when $k > \log n$. Similar arguments can be used for the case when $k \leq \log n$.

We have that $APP_k \leq APP'_k$, and we only need to prove that $APP'_k \leq O(\min\{k, r\} \cdot \log^2 n) \cdot P_{(k+1)/4})$. Using arguments similar to the ones in Theorem 3.1, we can show that if we apply the algorithm in [21]

on G' recursively (as we described in $MULTIWAY$ on G_h) we have that $APP'_k \leq O(\log^2 n) \cdot P'_{(k+1)/4}$. (Recall that the algorithm in [20] guarantees an approximation bound of $O(\log n)$ from the optimal balanced bipartition.) Thus, we only need to show that $OPT'_{(k+1)/4} \leq O(\min\{k, r\}) \cdot P_{(k+1)/4}$. Observe that there exists a partition of G', call it $APP''_{k/3}$, such that $APP''_{(k+1)/4} \leq \min\{k, r\} \cdot P_{(k+1)/4}$. This is true since the largest hyperedge on G_{h1} can connect at most $\min\{k, r\}$ nodes. However, $P'_k \leq APP''_k$ since P'_k is the optimal partition of G'. □

Let c_{max} be the maximum capacity on any edge of G_h and d_h be the maximum node degree of G_h. The theorem below can be shown with arguments as in [20].

Theorem 3.3 *Algorithm $MULTIWAY1$ can be implemented in $O(n^2 \cdot m \cdot \log(n^2/m) \cdot \log(n \cdot c_{max}))$ expected time. If G_h has uniform weights on its nodes then it can be implemented in $O(n \cdot d_h \cdot m \cdot \log^2 n)$ time.*

Similar results can also be derived for the case when we impose a lower bound of $k/3$ for each set. The justification is identical to the one presented for the algorithm in the previous section.

3.3 Planar hypergraphs

In this section, we consider the special case of partitioning a planar hypergraph $G_h = (V_h, E_h)$ into sets of size at most $k = O(\log n)$. As mentioned in the Introduction, we define a hypergraph G_h to be planar if its bipartite graph representation is planar. Our algorithm for planar hypergraph multiway partition uses the separator algorithm obtained in [22] for planar graphs. An initial a preliminary step in our multiway planar hypergraph partitioning algorithm is to transform the input hypergraph $G_h = (V_h, E_h)$ to a planar graph $G = (V, E)$, so that $V = V_h$ and each hyperedge $e \in E_h$ connecting nodes v_i, $1 \leq i \leq p$, corresponds to a chain connecting the same nodes in V. (Each edge in the chain has weight equal to the weight of the hyperedge. If parallel edges are formed between any two nodes in V, we consider them as distinct edges.) Clearly, G is planar. Let d_h and w_h denote the maximum node degree of G_h, and the maximum weight in any hyperedge $e \in E_h$. Similarly, let d and w denote the maximum node degree of G, and the maximum weight in any hyperedge $e \in E$. It can be easily observed that $d = d_h$ and $w = w_h$.

In this section, we will describe a multiway partitioning algorithm, we call it $PLANAR$, that guarantees that the sum of the weights in the returned partition on G is at most $O(\sqrt{w \cdot d} \cdot l(V)/\sqrt{\log n})$. This implies that the sum of the weights on the hyperedges cut by the same node partitioning of G_h is also $O(\sqrt{w_h \cdot d_h} \cdot l(V)/\sqrt{\log n})$.

Assume w.l.o.g. that graph G consists of one connected component, otherwise the procedure described below is applied to every connected component of G. Let P_k be the optimal partition (and its cost) of G_h into sets of size at most k. We have that $P_k \geq l(V)/k - 1 = O(l(V)/k)$. Assume now that the cost of the partition returned by $PLANAR$ is $O(\sqrt{w_h \cdot d_h} \cdot l(V)/\sqrt{\log n})$. Clearly, the cost of the partition is never more than $P_k + O(\sqrt{w_h \cdot d_h} \cdot l(V)/\sqrt{\log n})$, and from the above equalities we can quickly deduce that the cost is $O(1 + k \cdot \sqrt{d_h \cdot w_h}/\sqrt{\log n}) \cdot P_k$, which is the main result of this section.

Therefore it remains to show to describe algorithm $PLANAR$ on G, and show that the cost of the partition that it returns is $O(\sqrt{w_h \cdot d_h} \cdot l(V)/\sqrt{\log n})$. Algorithm $PLANAR$ is presented below.

Algorithm PLANAR

Step 1 Apply recursively the algorithm given in [22] until every resulting set is of size at most $\log n$. At each recursive call, we keep the removed edges in a variable named D.

Step 2 Search for connected components in each of the resulting sets.

Step 3 Apply an exponential algorithm on each connected component with size more than k, to end up with components with size at most k.

End of PLANAR

In order to show that the cost of the partition returned by $PLANAR$ is $O(\sqrt{w_h \cdot d_h} \cdot n/\sqrt{\log n})$, it suffices to show that the sum of the weights in the edges in D is $O(\sqrt{w \cdot d} \cdot n/\sqrt{\log n})$. This is shown in the following.

Assume that the numbering of the levels at the partition tree is bottom-up starting from $i := 0$. Let $|S|$ be the the number of sets at each level i in the partition tree created by $PLANAR$. Clearly, each set at level 1 has more than $\log n$ nodes. It follows that each set at level i, for each $i > 1$, has at least $(3/2)^{i-1} \cdot \log n$ nodes. Therefore

$$|S| \leq \frac{(2/3)^{i-1} \cdot l(V)}{\log n}. \tag{18}$$

Let a_1 ,..., a_p be p sets, each having size $l(V_j)$ and such that $\sum_{j=1}^{p} l(V_j) \leq n$. We observe that

$$\sum_{j=1}^{p} \sqrt{l(V_j)} \leq \sqrt{l(V) \cdot p}. \tag{19}$$

(19) is shown by squaring both its sides, rewriting it (after some calculations) as

$$\sum_{i,j \in \{1,..,p\} \wedge i \neq j} 2 \cdot \sqrt{l(V_i) \cdot l(V_j)} \leq \sum_{j=1}^{p} l(V_j) \cdot (p-1). \tag{20}$$

However, (20) is expressed as

$$0 \leq \sum_{j=1}^{p} l(V_j) \cdot (p-1) - \sum_{i,j \in \{1,..,p\} \wedge i \neq j} 2\sqrt{l(V_i) \cdot l(V_j)} \Leftrightarrow 0 \leq \sum_{i,j \in \{1,..,p\} \wedge i \neq j} (l(V_i) - l(V_j) + 2 \cdot \sqrt{l(V_i) \cdot l(V_j)}),$$

which is always satisfied since it can be expressed as

$$0 \leq \sum_{i,j \in \{1,..,p\} \wedge i \neq j} (\sqrt{l(V_i)} - \sqrt{l(V_j)})^2.$$

Let d_j be the maximum node degree among the nodes in the set V_j. Let d denote the maximum node degree among the nodes of G. Let n be the number of nodes of G, and p be the number of sets. Observe that $\sum_{j=1}^{p} \sqrt{l(V_j) \cdot d_j} \leq \sum_{j=1}^{p} \sqrt{l(V_j) \cdot d}$. By (19), we have that $\sum_{j=1}^{p} \sqrt{l(V_j) \cdot d} \leq \sqrt{l(V) \cdot p \cdot d}$, and that

$$\sum_{j=1}^{p} \sqrt{l(V_j) \cdot d_j} \leq \sqrt{n \cdot p \cdot d}. \tag{21}$$

We now use (18) and (21) to show that D is $O(l(V) \cdot \sqrt{d \cdot w}/\sqrt{\log n}$. We have the following lemma.

Lemma 3.3 *D is $O(\frac{l(V) \cdot \sqrt{d \cdot w}}{\sqrt{\log n}})$.*

Proof: Let a_i be a set of n_i nodes at level i of the partition tree that is partitioned by a recursive call of $PLANAR$. The size of D is expressed as

$$O(\sum_i \sqrt{n_i \cdot d_i \cdot w}).$$

By (18), we rewrite the latter as

$$O(\sum_{p=1}^{\log(\frac{l(V)}{\log n})(\frac{2}{3})^{p-1}\cdot\frac{l(V)}{\log n}}\sum_{j=1}\sqrt{n_j\cdot d\cdot w}),$$

where $l(V_j) \leq l(V) \wedge l(V_j) > 0$.
Using (21), we have that

$$O(\sum_{p=1}^{log(\frac{l(V)}{\log n})}\sqrt{\frac{l(V)\cdot d\cdot n\cdot w}{\log n}\cdot(\frac{2}{3})^{p-1}}) \leq O(\sum_{p=1}^{\infty}\sqrt{\frac{n\cdot d\cdot w\cdot l(V)}{\log n}\cdot(\frac{2}{3})^{p-1}}). \tag{22}$$

If we set i = p - 1, (22) becomes

$$O(\frac{l(V)\cdot\sqrt{w\cdot d}}{\sqrt{\log n}})\cdot\sum_{i=0}^{\infty}(\frac{2}{3})^{\frac{i}{2}} = O(\frac{l(V)\cdot\sqrt{d\cdot w}}{\sqrt{\log n}}), \tag{23}$$

which proves the lemma. □

We use Lemma 3.3 to obtain the approximation bound of *PLANAR*.

Theorem 3.4 *If $k \leq \log n$, then the cost of the partition is $O(1+\frac{k\cdot\sqrt{w_h\cdot d_h}}{\sqrt{\log n}})\cdot P_k$, where P_k is the optimal partition of G_h. Moreover, if $k \leq \sqrt{\log n}$, $d_h = O(1)$ and the hyperedges have uniform weights, then the cost is $O(1+\epsilon)\cdot P_k$, for some $\epsilon < 1$.*

The same approximation bound holds even if we insist that the size of a set is never less than $k/3$, and subject to the upper bound restriction described earlier. Furthermore, it is easy to observe that the time complexity of the described algorithm is only $O(n\cdot\log n)$. This is justified since the algorithm in [22] has $O(n)$ time complexity, and the depth of the partition tree is $O(\log(l(V)/k)) = O(\log n)$.

4 Conclusions

We presented approximation algorithms for balanced bipartitioning and multiway partitioning problems into bounded size sets on hypergraphs. The bounds are within polylogarithmic factors to the optimal. Most of the presented techniques use concurrent flow arguments. The presented approximation algorithms can be combined with existing heuristics to improve the cost of the obtained partitions. Extensive experimentation will then show whether provably good initial solutions, obtained as shown in this paper, trap us to local minima when used in established heuristics as in [5, 12].

The following are interesting open problems:

(a) Improve the presented approximation bounds using flow or other techniques.

(b) Improve the time complexity of the presented algorithms.

(c) Obtain polynomial time approximation algorithms for the *bisection* problem which finds numerous applications in VLSI. The goal is to partition G_h into two equal sets so that the number of interconnecting hyperedges (or the sum of their weights) is minimized. Approximation algorithms for hypergraph bisection have been presented only for special cases of hypergraphs [29]. We are not aware of any polynomial time approximation algorithm that applies to any hypergraph G_h.

References

[1] M. Abramovici, M.A. Breuer, and A.D. Friedman. Digital Systems Testing and Testable Design. Computer Science Press, 1990.

[2] T. Bui, C. Heigham, C. Jones and T. Leighton. Improving the performance of the Kernighan-Lin and simulated annealing graph bisection algorithm. In *Proceedings of the* 25^{th} *ACM/IEEE Design Automation Conference*, 1988.

[3] E. R. Barnes, A. Vannelli and J. Q. Walker. A new heuristic for partitioning the nodes of a graph. *SIAM Journal of Discrete Mathematics*, vol.1, no.3, pages 299–305, August 1988.

[4] C–I. H. Chen, J. Yuen, and J-D. Lee, "Autonomous- Tool for hardware partitioning in a Built-In Self-Test environment", in the Proceedings of the 1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors, pp. 264 - 267, October 11 - 14, 1992, Cambridge, Massachussets.

[5] C. M. Fiduccia and R. M. Mattheyses. A linear time heuristic for improving network partitions. In *Proceedings of the* 19^{th} *ACM/IEEE Design Automation Conference*, pages 175–181, 1982.

[6] A. V. Goldberg. *Personal communication*, January 1991.

[7] M.D. Grigoriadis and L.G. Khachiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. DIMACS, Technical Report 91-24, March 1991.

[8] M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness.* Freeman, San Francisco, 1979.

[9] N. Garg, V.V. Vazirani, M. Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. In the DIMACS workshop on approximation algorithms for combinatorial optimization, 24 - 26 March, 1993, Rutgers University, New Jersey.

[10] J. E. Hopcroft and R. E. Tarjan. Efficient planarity testing. *Journal of ACM*, vol. 21, pages 549–568, 1974.

[11] M. Iri. On an extension of the maximum-flow minimum cut theorem to multicommodity flows. *Journal of Operations Research Society of Japan*, vol. 13, no.3, pages 129–135, January 1971.

[12] B. W. Kernighan and S. Lin. An efficient procedure for partitioning graphs. *Bell Systems Technical Journal*, vol. 49, no. 2, February 1970.

[13] P. Klein, A. Agrawal, R. Ramamurthy and S. Rao. Approximation through multicommodity flow. In *Proceedings of the* 31^{st} *Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 1990.

[14] P. Klein, S. Plotkin, C. Stein, and E. Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts, Technical Report 961, School of Operations Research and Industrial Engineering, Cornell University, 1991. A preliminary version, entitled Leighton-Rao might be practical: faster approximation algorithms for concurrent flow with uniform capacities, appears in the *Proceedings of the 22st Annual ACM Symposium on Theory of Computing*, May 1990.

[15] J.J. Hallenbeck, N. Kanopoulos and J.R. Cybrinski. The Test Engineer's Assistant: A design environment for testable and diagnosable systems, IEEE Transactions on Industrial Electronics, special issue on Electronic testing, May 1989.

[16] S. Kapoor and P. M. Vaidya. Fast algorithms for convex quadratic programming and multicommodity flows. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 147–159, 1986.

[17] E.L. Lawler. Combinatorial Optimization: Algorithms and Matroids. Holt, Rinehart and Winston, 1976.

[18] T. Lengauer. *Combinatorial algorithms for integrated circuit layouts.* Teubner/Wiley series of applicable theory in computer science, N.Y., 1990.

[19] T. Leighton, F. Makedon and S. Tragoudas. Approximation algorithms for VLSI partitioning problems. In Proceedings of the 1990 International Symposium on Circuits and Systems, pages 2865–2869, May 1-3, New Orleans.

[20] T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos and S. Tragoudas. Fast approximation algorithms for multicommodity flow problems. In *Proceedings of the* 23^{th} *Annual ACM Symposium on Theory of Computing*, pages 101–112, New Orleans, May 1991.

[21] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 422–431. IEEE, October 1988.

[22] G. Miller. Finding small simple cycle separators for 2-connected planar graphs. In *Proceedings of the* 16^{th} *Symposium on the Theory of Computing*, pages 376–382, 1984.

[23] F. Makedon and S. Tragoudas. Approximating the minimum net expansion: Near optimal solutions to circuit partitioning problems. In Proceedings of the 1990 Workshop on Graphtheoretic concepts in computer science, June 19–22, Berlin, Germany.

[24] S. Plotkin, and E. Tardos. Improved bounds on the max-flow min-cut ratio for multicommodity flows. In the 1993 DIMACS workshop on approximation algorithms for combinatorial optimization, March 24-26, 1993, Rutgers University, New Jersey.

[25] S. B. Rao. Finding near optimal separators in planar graphs. In *Proceedings of the 28^{th} Annual IEEE Symposium on Foundations of Computer Science*, pages 225–237, 1987.

[26] S. B. Rao. Algorithms for finding small edge cuts in planar graphs. In the Proceedings of the 24^{th} Annual ACM Symposium on Theory Of Computing (STOC'92), pp. 229 - 240, May 4 - 6, 1992, Vicoria B.C., Canada.

[27] F. Shahroki and D. W. Matula. The maximum concurrent flow problem. *Journal of the ACM*, vol. 37, pages 318–334, 1990.

[28] P.R. Suaris, and G. Kedem. A quadrasection-based combined based place and route scheme for standard cells. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. CAD vol. 8, no. 3, pp. 234 - 244, 1989.

[29] S. Tragoudas. VLSI partitioning approximation algorithms using multicommodity flow and other techniques. *Ph.D. Thesis,* The University of Texas at Dallas, Richardson, Texas 75083-0688, June 1991.

[30] S. Tragoudas, F. Makedon, R. Farell. "Circuit partitioning into small sets: A tool to support testing with further applications". In the Proceedings of the 2^{nd} European Design Automation Conference, pp. 518 - 523, 25 - 28 February, 1991, Amsterdam, The Netherlands.

[31] P. M. Vaidya. Speeding up linear programming using fast matrix multiplication. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 332–337, 1989.