Flows, Cuts and Integral Routing in Graphs - an Approximation Algorithmist's Perspective

Julia Chuzhoy [*]

Abstract. Flow, cut and integral graph routing problems are among the most extensively studied in Operations Research, Optimization, Graph Theory and Computer Science. We survey known algorithmic results for these problems, including classical results and more recent developments, and discuss the major remaining open problems, with an emphasis on approximation algorithms.

Mathematics Subject Classification (2010). Primary 68Q25; Secondary 68Q17, 68R05, 68R10.

Keywords. Maximum flow, minimum cut, network routing, approximation algorithms, hardness of approximation, graph theory.

1. Introduction

In this survey we consider flow, cut, and integral routing problems in graphs. These three types of problems are among the most extensively studied in Operations Research, Optimization, Graph Theory, and Computer Science. Problems of these types naturally arise in many applications, and algorithms for solving them are among the most valuable and powerful tools in algorithm design and analysis.

In the classical maximum s–t flow problem, we are given an n-vertex graph $G = (V, E)$, that can be either directed or undirected, with non-negative capacities $c(e)$ on edges $e \in E$, and two special vertices: s, called the source, and t, called the destination. Let $\mathcal{P}$ be the set of all paths connecting s to t in G. An s–t flow f is an assignment of non-negative values $f(P)$ to all paths $P \in \mathcal{P}$, such that for each edge $e \in E$, the flow through e does not exceed its capacity $c(e)$, that is, $\sum_{P:e\in P} f(P) \leq c(e)$. The value of the flow f is $\sum_{P\in\mathcal{P}} f(P)$, and the goal is to find a flow of maximum value. The maximum flow problem was introduced in the 50's in order to model the capacity of the Soviet and East European railway systems. Ford and Fulkerson [FF62] were the first to provide an efficient algorithm for solving the problem. The problem can be expressed as a linear program (LP):

[*]Supported in part by NSF grant CCF-1318242.

$$\begin{array}{llll}
\text{(LP-flow)} & \max & \sum_{P\in\mathcal{P}} f(P) & \\
& \text{s.t.} & & \\
& & \sum_{P:e\in P} f(P) \le c(e) & \forall e\in E \\
& & f(P)\ge 0 & \forall P\in\mathcal{P}
\end{array}$$

So far, in our definition of the maximum s–t flow problem, the number of paths P with non-zero flow value $f(P)$ may be exponentially large in the graph size, and so can the number of variables of (LP-flow). Fortunately, there is an equivalent "compact" LP-formulation of the problem, whose solution can be efficiently converted into a solution to (LP-flow), where the number of paths P with $f(P) > 0$ is bounded by $|E|$. This provides an efficient algorithm to solve (LP-flow), as long as we are only required to list the non-zero values $f(P)$ in the solution.

A very useful feature of the maximum s-t flow problem is that, if all edge capacities $c(e)$ are integral, then there is a maximum flow where for each $P \in \mathcal{P}$, $f(P)$ is integral, and such a flow can be found efficiently. This property is often referred to as the *integrality of flow.* In particular, if all edge capacities are unit, then we can efficiently find a maximum-cardinality collection $\mathcal{P}'$ of paths connecting s to t, such that the paths in $\mathcal{P}'$ are *edge-disjoint*: that is, every edge of G belongs to at most one path of $\mathcal{P}'$.

A problem closely related to maximum s–t flow is minimum s–t cut. The input to this problem is the same as the input to the maximum s–t flow problem, only now we will think of the values $c(e)$ as edge costs and not capacities. The goal is to select a minimum-cost subset $E' \subseteq E(G)$ of edges, such that $G \setminus E'$ contains no path connecting s to t, where the cost of E' is $\sum_{e\in E'} c(e)$. It is easy to see that the value of the maximum s–t flow cannot exceed the value of the minimum s–t cut in any graph: every path $P \in \mathcal{P}$ must contain at least one edge of E', and so the total flow carried by the paths in $\mathcal{P}$ cannot exceed the total capacity of the edges in E'. The classical result of Ford and Fulkerson [FF62], often referred to as the Max-Flow Min-Cut Theorem, shows that the opposite is also true, that is, in any graph, the value of the minimum s–t cut is equal to the value of the maximum s–t-flow! In fact, their algorithm for computing maximum flow can also be used to compute a minimum cut. Therefore, we can see minimum cut as revealing the bottleneck in the routing capacity of a graph: if the maximum amount of flow that can be sent from s to t is x, then we can produce a certificate for this fact in the form of a valid flow of value x, and an s–t cut of cost x. A convenient way of seeing the connection between flows and cuts is by computing the dual linear program of (LP-flow), that we will call (LP-cut) for reasons that will become apparent below.

$$\begin{array}{lllll}
\text{(LP-cut)} & \min & \sum_{e\in E} c_e x_e & & \\
& \text{s.t.} & & & \\
& & \sum_{e\in P} x_e \ge 1 & \forall P\in\mathcal{P} & (1) \\
& & x_e \ge 0 & \forall e\in E & (2)
\end{array}$$

Let us start by adding the following *integrality* constraints to (LP-cut):

$$x_e \in \{0,1\} \qquad \forall e \in E \tag{3}$$

This combination of a linear program with integrality constraints is called *integral linear program*, and we denote it by (ILP-cut). It is immediate to see that (ILP-cut) is equivalent to the minimum s–t cut problem: we set $x_e = 1$ if e belongs to the solution E', and $x_e = 0$ otherwise. Constraint (1) ensures that every path from s to t contains at least one edge from E' - that is, $G \setminus E'$ contains no s–t path. Of course, any feasible solution of (ILP-cut) is also a feasible solution to (LP-cut). However, (LP-cut) allows more solutions: for example, solutions where the variables x_e take fractional values. We say that (LP-cut) is a *relaxation* of the minimum s–t cut problem. The optimal solution to (LP-cut) is called the optimal *fractional* solution to minimum s–t cut, and its value is denoted by $\mathsf{OPT}_{\mathsf{LP}}$. The optimal solution to (ILP-cut) is called the optimal *integral* solution; its value is denoted by OPT, and it is also the value of the minimum s–t cut in G. Since the optimal integral solution is a valid solution to (LP-cut), $\mathsf{OPT}_{\mathsf{LP}} \leq \mathsf{OPT}$ must hold. Interestingly, in this particular linear program, the two values are equal. Moreover, there is an efficient algorithm, that, given a fractional solution to (LP-cut), computes a valid integral solution of the same value.

We describe the algorithm for directed graphs; the algorithm for undirected graphs is similar, with minor adjustments. The idea is to view the values x_e in the optimal solution of (LP-cut) as edge lengths. We can then define, for every pair (u,v) of vertices, the distance $d(u,v)$ from u to v, to be the length of the shortest path connecting u to v, under the edge lengths x_e. Constraint (1) ensures that $d(s,t) \geq 1$, and it is easy to see that for every edge $e=(u,v)$:

$$d(s,v) \leq d(s,u) + x_e \tag{4}$$

Let us choose a value $\rho \in (0,1)$ uniformly at random, and let $B(s,\rho) = \{v \mid d(s,v) \leq \rho\}$ be the ball of radius ρ around s. This ball defines an s–t cut $E'_\rho \subseteq E$, where $e=(u,v) \in E'_\rho$ if $u \in B(s,\rho)$ and $v \notin B(s,\rho)$. The probability that $e=(u,v)$ belongs to E'_ρ is the probability that ρ lies between $d(s,u)$ and $d(s,v)$, which, from (4), is bounded by x_e. The expected cost of the cut E'_ρ is then:

$$\sum_{e \in E} c(e) \cdot \mathbf{Pr}_\rho\left[e \in E'_\rho\right] \leq \sum_{e \in E} c(e) x_e = \mathsf{OPT}_{\mathsf{LP}}.$$

At least one value $\rho : 0 < \rho < 1$ must satisfy $\sum_{e \in E'_\rho} c_e \leq \mathsf{OPT}_{\mathsf{LP}}$. We can find this value by going over all possible values of ρ and computing E'_ρ for each of them. Fortunately, the number of different values of ρ that we need to check is not very large - it is enough to consider all values in set $\{d(s,v) \mid v \in V\}$.

An algorithm that, given a fractional solution to a linear program, computes an integral solution is called an *LP-rounding algorithm.* If the value of the solution produced by the algorithm equals to the value of the fractional solution, then this algorithm can be used to solve the problem exactly. If additionally the

LP-rounding algorithm is efficient, then we obtain an efficient algorithm for the problem, thus proving that it is in **P**. This is exactly what we have just shown for minimum s–t cut. However, many optimization problems that we consider in this survey are **NP**-hard, and therefore we do not expect them to have efficient algorithms. Instead, we will often look for *approximation algorithms* - efficient algorithms that solve the problem approximately. Given a minimization problem Π, we say that an efficient algorithm $\mathcal{A}$ is an α-approximation algorithm for Π, if for any instance I of Π, algorithm $\mathcal{A}$ produces a solution of value at most $\alpha \cdot \mathsf{OPT}(I)$, where $\mathsf{OPT}(I)$ is the value of the optimal solution for I. For a maximization problem, an α-approximation algorithm needs to produce a solution of value at least $\mathsf{OPT}(I)/\alpha$. Different optimization problems often have different approximation factors achievable by efficient algorithms. The approximation factor α may be a constant, or some function of the input size n (like $O(\log n)$, $O(\sqrt{n})$, and so on). For many optimization problems, we still do not know what is the best approximation factor α^* achievable for them. In order to determine this factor, in addition to designing an approximation algorithm, that establishes an upper bound on α^*, we need to provide a lower bound on α^*. This is usually done by proving *hardness of approximation*, or *inapproximability* results: namely, that achieving a better than α^*-approximation for a given problem Π is an **NP**-hard problem.

As we have shown, there is an efficient LP-rounding algorithm for minimum s–t cut that can be used, together with (LP-cut), to solve the problem exactly. For many other minimization problems, the value of the integral solution produced by an LP-rounding algorithm for a minimization problem is greater than $\mathsf{OPT}_{\mathsf{LP}}$. However, if the value of the solution is at most $\alpha \cdot \mathsf{OPT}_{\mathsf{LP}}$ for any input instance I, then, since $\mathsf{OPT}_{\mathsf{LP}} \leq \mathsf{OPT}$ we obtain an α-approximate LP-rounding algorithm. The technique of rounding linear programming relaxations is one of the most powerful and widely used tools in the design of approximation algorithms.

From the strong duality theorem, the optimal value of (LP-flow) equals to the optimum value of (LP-cut), that is, maximum flow equals to the value of the minimum fractional cut. But since the values of the optimal fractional and the optimal integral solutions to the s–t cut problem are the same, we get that the maximum flow value equals to the value of the minimum cut in any graph G.

Linear program (LP-cut) is one of the rare cases where the optimal fractional and the optimal integral solutions have the same value. For many other minimization problems and their linear programming relaxations, $\mathsf{OPT}_{\mathsf{LP}} < \mathsf{OPT}$ holds. Given a minimization problem Π, and a linear programming relaxation (LP-rel) for Π, the *integrality gap* of (LP-rel) is the largest possible ratio between the value OPT of the optimal integral solution and the value $\mathsf{OPT}_{\mathsf{LP}}$ of the optimal fractional solution, achieved by any instance I of Π. (For maximization problems we reverse the ratio, and the integrality gap is the maximum of $\mathsf{OPT}_{\mathsf{LP}}/\mathsf{OPT}$ over all instances; so the integrality gap is always at least 1). If the integrality gap of a linear programming relaxation (LP-rel) of problem Π is α, then no LP-rounding algorithm can achieve a better than α-approximation for the problem. This statement however is only true for the specific linear programming relaxation (LP-rel) of Π. Often one can come up with different linear programming relaxations of the same problem, that have different integrality gaps, and LP-rounding algorithms achieving different approximation factors. Studying integrality gaps of linear programs is therefore crucial in understanding the power and the limitations of the LP-rounding approach for specific optimization problems. Often, instances exhibiting large LP-integrality gaps can give us insight into the structure of hard instances of the problem, and this can help us prove inapproximability results. Alternatively, they can help us strengthen the LP relaxation and obtain better LP-rounding algorithms.

As we have already seen, the integrality ratio of (LP-cut) is 1. We have also already mentioned that, if all edge capacities are integral, then there is an optimal solution to the maximum s–t flow problem where all values $f(P)$ are also integral. Therefore, if all edge capacities are integral, then the integrality gap of (LP-flow) is also 1. In the following sections we consider generalizations of the maximum s–t flow problem, where instead of one source-destination pair, there are several such pairs. There are two natural ways to define the objective function in this setting: we can try to maximize the total amount of flow sent between all source-destination pairs - a problem known as the maximum multicommodity flow; or we can try to maximize a value λ such that all demand pairs can simultaneously send λ flow units between them - this is known as maximum concurrent flow. We define the two corresponding graph cut problems, minimum multicut and sparsest cut, and study their LP-relaxations, as well as known approximation algorithms and hardness results in Sections 2 and 3. Unfortunately, the integrality of flow does not hold anymore in the multiple source-destination pairs setting, and the problem of computing maximum integral flow becomes **NP**-hard. We discuss approximation algorithms and hardness results for integral routing problems in Sections 4 and 5.

Before we proceed, let us mention another common and useful version of the maximum s–t flow problem, where the capacities are on the graph vertices and not on edges. In this problem, we are given a graph $G = (V, E)$, a source vertex $s \in V$ and a destination vertex $t \in V$, and capacity values $c(v)$ for all vertices $v \in V \setminus \{s, t\}$. As before, let $\mathcal{P}$ denote the set of all paths connecting s to t in G. A valid flow assigns values $f(P) \in \mathbb{R}^+$ to each path $P \in \mathcal{P}$, so that for every vertex $v \in V \setminus \{s, t\}$, $\sum_{P: v \in P} f(P) \leq c(v)$. In the corresponding vertex cut problem, we are given costs $c(v)$ on vertices $v \in V$, and the goal is to select a minimum-cost subset $S \subseteq V \setminus \{s, t\}$ of vertices, so that $G \setminus S$ contains no path connecting s to t. The node-capacitated version of the maximum flow problem behaves very similarly to the edge-capacitated one. We can write a linear programming relaxation, similar to (LP-flow), which can be solved efficiently using similar methods. The dual of this linear program is a relaxation of the minimum vertex cut problem. As in the edge-capacitated version of the problem, the integrality gap of the LP-relaxation for minimum vertex cut is 1, and, when all vertex capacities are integral, the integrality gap of the LP-relaxation for node-capacitated maximum flow is also 1. The maximum flow value and the minimum cut value are therefore equal for any graph G even in this model. If all vertex capacities are unit, then we obtain an efficient algorithm for computing a maximum-cardinality set $\mathcal{P}'$ of internally node-disjoint paths connecting s to t (so every vertex $v \in V \setminus \{s, t\}$ may belong to at most one path of $\mathcal{P}'$). Therefore, the cardinality of $\mathcal{P}'$ is equal to the value of the minimum vertex s–t cut in any graph G - this is known as Menger's theorem [Men27].

2. Maximum Multicommodity Flow and Minimum Multicut

A natural generalization of the maximum s–t flow problem is maximum multicommodity flow. In this problem, instead of a single source-destination pair (s,t), we are given a collection of k such pairs $\{(s_1,t_1),\ldots,(s_k,t_k)\}$, that we call *demand pairs*. The goal is to send maximum amount of flow between the demand pairs, without violating the edge capacities: that is, for each $1 \leq i \leq k$, the flow leaving s_i must arrive at t_i, and the total amount of flow traversing any edge e is at most $c(e)$. It is sometimes convenient to think of having k different flow types, or *commodities*, where the ith commodity needs to be sent from s_i to t_i. For each $1 \leq i \leq k$, let $\mathcal{P}_i$ denote the set of all paths connecting s_i to t_i in G. The following linear program is a generalization of (LP-flow) to the multi-commodity setting:

$$
\begin{array}{lll}
\text{(LP-multi-flow)} & \max & \sum_{i=1}^k \sum_{P\in\mathcal{P}_i} f(P) \\
 & \text{s.t.} & \\
 & & \sum_{P:e\in P} f(P) \leq c(e) \quad \forall e\in E \\
 & & f(P) \geq 0 \qquad \forall 1\leq i\leq k \quad \forall P\in\mathcal{P}_i
\end{array}
$$

Like (LP-flow), this linear program can be solved efficiently using similar methods. The cut counterpart of maximum multicommodity flow is minimum multicut. In this problem, the input is the same as in the maximum multicommodity flow problem, but we view the values $c(e)$ as edge costs, rather than capacities. The goal is to select a minimum-cost subset $E' \subseteq E$ of edges, such that in graph $G \setminus E'$, there is no path connecting any source s_i to its destination t_i. As in the single-commodity scenario, it is easy to see that the value of the maximum multicommodity flow cannot exceed the value of the minimum multicut in any graph G, since for each $1 \leq i \leq k$, every path $P \in \mathcal{P}_i$ must contain at least one edge of E'. Therefore, the total amount of flow carried by the paths in $\bigcup_{i=1}^k \mathcal{P}_i$ is bounded by the total capacity of the edges in E'. The dual linear program of (LP-multi-flow) also happens to be a relaxation of minimum multicut:

$$
\begin{array}{llll}
\text{(LP-multicut)} & \min & \sum_{e\in E} c(e)x_e & \\
 & \text{s.t.} & & \\
 & & \sum_{e\in P} x_e \geq 1 \quad \forall 1\leq i\leq k \quad \forall P\in\mathcal{P}_i & (5) \\
 & & x_e \geq 0 \qquad \forall e\in E & (6)
\end{array}
$$

Indeed, if we restrict the values x_e to be in $\{0,1\}$, and let E' be the set of all edges e with $x_e = 1$, then Constraint (5) ensures that every path connecting any source s_i to its destination t_i contains at least one edge of E', or, equivalently, $G \setminus E'$ contains no path connecting s_i to t_i, for any $1 \leq i \leq k$. Let $\mathsf{OPT}_{\mathsf{LP}}$ be the value of the optimal solution to (LP-multicut), that we also call the *minimum fractional multicut value*. The optimal solution to the minimum multicut problem is denoted by OPT, and is called the *minimum integral multicut value*. From the LP-duality theorem, the value of the maximum multicommodity flow equals to the value of the minimum fractional multicut. However, the integrality gap of (LP-multicut) is no longer 1, and so the maximum multicommodity flow value may be smaller than the value of minimum multicut. The equality between maximum flow and minimum cut therefore breaks down in the multicommodity setting. However, we can still hope to obtain an approximate version of the Max-Flow Min-Cut Theorem, by bounding what is called the *flow-cut gap* - the largest possible ratio between maximum multicommodity flow and minimum multicut in any graph. Since the maximum multicommodity flow value equals to the value of the minimum fractional multicut, the flow-cut gap is precisely the integrality gap of (LP-multicut).

For undirected graphs, Garg, Vazirani and Yannakakis [GVY96], building on the work of Leighton and Rao [LR99] and Klein et al. [KARR90] showed that the integrality gap of (LP-multicut) is $O(\log k)$, by providing an efficient LP-rounding algorithm, whose approximation factor is $O(\log k)$. This bound on the integrality gap is almost tight: there is an instance of the minimum multicut problem, for which $\mathsf{OPT} = \Omega(\log k) \cdot \mathsf{OPT}_{LP}$ [LR99]. The integrality gap of (LP-multicut), and the flow-cut gap for undirected graphs are therefore well understood (to within a constant factor), and stand on $\Theta(\log k)$. The question of whether one can obtain a better than $O(\log k)$-approximation algorithm for undirected multicut by other methods, or perhaps by LP-rounding of a different linear programming relaxation remains wide open. The best currently known hardness of approximation result shows that for some constant c, the problem does not have a c-approximation algorithm, assuming that $\mathbf{P} \neq \mathbf{NP}$ [DJP+94]. Under a complexity assumption called the Unique Games Conjecture [Kho02], the undirected multicut problem is hard to approximate to within any constant factor [CKK+06, KV05]. The status of the Unique Games Conjecture is however still wide open.

The situation is very different in directed graphs. It is easy to obtain a factor k-approximation to the minimum multicut problem, by computing, for each $1 \leq i \leq k$, a minimum s_i–t_i cut E'_i, and returning $\bigcup_{i=1}^{k} E'_i$ as the solution to minimum multicut. Surprisingly, a beautiful construction of Saks et al. [SSZ04] shows that this algorithm is close to the best one can achieve via the LP-rounding of (LP-multicut), since the integrality gap of (LP-multicut), and hence the flow-cut gap, can be as large as $k-\epsilon$ for any $\epsilon > 0$ in directed graphs. The number of pairs k in their construction is however quite small when compared to the total number of vertices n in the graph: $k = \Theta(\log n / \log\log n)$, and hence, as a function of n, the lower bound they achieve on the integrality gap is only $\Omega(\log n / \log\log n)$. Unfortunately, [CK09] have shown that the integrality gap of (LP-multicut), and therefore the flow-cut gap in directed graphs is at least $\Omega(n^{1/7} / \operatorname{poly}\log n)$[1]. The best current approximation algorithm achieves an $O(n^{11/23} \cdot \operatorname{poly}\log n)$-approximation via LP-

[1]We say that $f(n) = \operatorname{poly}\log n$ if there is some constant c, such that $f = \Theta((\log n)^c)$.

rounding of (LP-multicut), thus providing an upper bound of $O(n^{11/23} \cdot \text{poly} \log n)$ on the flow-cut gap [CKR05, Gup03, AAC07]. The value of the flow-cut gap for directed graphs therefore remains open, but, unlike undirected graphs, it is polynomially large in n. Minimum multicut in directed graphs is hard to approximate to within factor $2^{\Omega(\log^{1-\epsilon} n)}$ for any constant $\epsilon > 0$, under the plausible complexity assumption that some problems in **NP** do not have efficient randomized algorithms [CK09].

3. Concurrent Flow and Sparsest Cut

Maximum concurrent flow problem can be seen as multicommodity flow with additional fairness requirements. The input to this problem is the same as in maximum multicommodity flow, but instead of routing maximum amount of flow between all demand pairs, we would like to ensure that **every** demand pair routes a significant amount of flow, and we measure our success by the smallest amount of flow routed between any demand pair. In other words, we would like to maximize a value λ, such that each demand pair (s_i, t_i) can route λ flow units from s_i to t_i simultaneously, and the total flow on any edge e does not exceed its capacity $c(e)$. The linear programming formulation of this problem uses the same notation as in (LP-multi-flow), and is as follows:

$$\begin{array}{llll}
\text{(LP-concurrent-flow)} & \max & \lambda & \\
 & \text{s.t.} & & \\
 & & \sum_{P\in\mathcal{P}_i} f(P) \geq \lambda & \forall 1 \leq i \leq k \\
 & & \sum_{P:e\in P} f(P) \leq c(e) & \forall e \in E \\
 & & f(P) \geq 0 & \forall 1 \leq i \leq k \quad \forall P \in \mathcal{P}_i
\end{array}$$

Often, a more general version of this problem is considered, where each demand pair (s_i, t_i) is associated with a demand value $D_i \geq 0$, and we need to route λD_i flow units from s_i to t_i simultaneously, without violating the edge capacities, for largest possible value λ. Linear program (LP-concurrent-flow) can also be solved efficiently using methods similar to those discussed in Section 1. The dual linear program for (LP-concurrent flow), that we call (LP-spcut), appears below.

$$\begin{array}{lllll}
\text{(LP-spcut)} & \min & \sum_{e\in E} c(e)x_e & & \\
 & \text{s.t.} & & & \\
 & & \sum_{e\in P} x_e \geq h_i & \forall i : 1 \leq i \leq k, \forall P \in \mathcal{P}_i & (7) \\
 & & \sum_{i=1}^{k} h_i \geq 1 & & (8) \\
 & & x_e \geq 0 & \forall e \in E & \\
 & & h_i \geq 0 & \forall 1 \leq i \leq k &
\end{array}$$

This linear program can be seen as a relaxation of a different graph cut problem, called the sparsest cut problem. Suppose we are given a graph $G = (V, E)$ with costs $c(e)$ on edges $e \in E$, and a collection $\mathcal{M} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ of demand pairs. For any subset $S \subseteq V$ of vertices, let $\overline{S} = V \setminus S$. Let $E(S, \overline{S})$ denote the set of all edges with exactly one endpoint in S, and let $D(S, \overline{S})$ be the set of all demand pairs (s_i, t_i), where exactly one of s_i, t_i belongs to S. The *sparsity* of S is $\frac{\sum_{e \in E(S,\overline{S})} c(e)}{|D(S,\overline{S})|}$. In the sparsest cut problem, the goal is to find a subset $S \subseteq V$ of vertices of minimum sparsity. If the set $\mathcal{M}$ of the demand pairs contains every pair of vertices of G, then the problem is called *uniform sparsest cut*. The general version of the problem, where $\mathcal{M}$ can be arbitrary, is often called the *non-uniform sparsest cut* problem.

Sparsest cut is one of the central combinatorial optimization problems. It is closely related to the important graph theoretic notions of graph expansion and graph conductance. Approximation algorithms for the sparsest cut problem are often used as subroutines in algorithms for problems arising in many different areas of Computer Science. As an example, one of the most useful paradigms in algorithm design is divide-and-conquer, that often requires a small balanced partition of a given graph G. That is, we need to partition $V(G)$ into two subsets V_1, V_2, each of which only contains a constant fraction (say at most 2/3) of the vertices of G, such that the number of edges $|E(V_1, V_2)|$ is minimized. This problem can be approximately solved by using an approximation algorithm for the sparsest cut problem as a subroutine.

In order to see that (LP-spcut) is a relaxation of the sparsest cut problem, consider any solution S to the sparsest cut problem, and let $E' = E(S, \overline{S})$. For each edge $e \in E$, define a new variable x'_e whose value is 1 if $e \in E'$ and 0 otherwise. For each $i : 1 \leq i \leq k$, define a new variable h'_i, whose value is 1 if $(s_i, t_i) \in D(S, \overline{S})$, and 0 otherwise. Let $D = |D(S, \overline{S})|$. We are now ready to define a solution to (LP-spcut): for each edge $e \in E$, set $x_e = x'_e/D$, and for each $1 \leq i \leq k$, set $h_i = h'_i/D$. It is then easy to see that we have defined a feasible solution to the linear program (LP-spcut), and the value of the solution $\sum_e c(e)x_e = \frac{\sum_{e \in E} c(e)x'_e}{D} = \frac{\sum_{e \in E'} c(e)}{|D(S,\overline{S})|}$ is exactly the sparsity of S. As with undirected multicut, there is an LP-rounding approximation algorithm for the sparsest cut problem, whose approximation factor is $O(\log k)$ in undirected graphs [LR99, LLR95, AR98], and a matching lower bound of $\Omega(\log k)$ on the integrality gap of (LP-spcut) [LR99]. Therefore, the flow-cut gap between maximum concurrent flow and sparsest cut in undirected graphs is $\Theta(\log k)$. In a major breakthrough, Arora, Rao and Vazirani [ARV09] designed an $O(\sqrt{\log n})$-approximation algorithm for uniform sparsest cut, by rounding a semidefinite relaxation of the problem. Their algorithm was later generalized to the non-uniform sparsest cut problem, where the approximation ratio becomes $O(\sqrt{\log k} \cdot \log \log k)$ [ALN05]. Somewhat surprisingly, these techniques do not seem to help with the minimum multicut problem, where the best approximation ratio still stands on $O(\log k)$, and is achieved by an LP-rounding algorithm of [LR99, GVY96]. On the negative side, it is known that the sparsest cut problem does not have a factor c-approximation for some specific constant c, unless all problems in **NP** have randomized subexponential time algorithms [AMS07], and this holds even for the uniform sparsest cut problem. Assuming the Unique Games Conjecture, the non-uniform sparsest cut is hard to approximate to within any constant factor [CKK$^+$06, KV05]. The approximability of the sparsest cut problem remains one of the central open questions in the area of approximation algorithms. Some progress has recently been made on special cases of the problem [GS11, AGS13].

For directed graphs, the notion of a sparsest cut can be defined in two distinct ways. In one version of the problem, which we refer to as the *bipartite sparsest cut*, the sparsest cut in a graph is a bipartition of vertices into two sets S and $\bar{S}$ that minimizes the ratio of $\frac{\sum_{e\in|E(S,\overline{S})|} c(e)}{|D(S,\overline{S})|}$. In the second version, which we refer to as the *non-bipartite sparsest cut*, we need to select a subset E' of edges, minimizing the ratio of $\sum_{e\in E'} c(e)$ to the number of the demand pairs disconnected in $G\setminus E'$. We note that (LP-spcut) is a relaxation of the non-bipartite sparsest cut. In undirected graphs, it is easy to see that the two notions are equivalent, but this is not the case in directed graphs. The best currently known approximation ratio for the non-bipartite sparsest cut is $O(n^{11/23}\operatorname{poly}\log n)$, achieved by LP-rounding of (LP-spcut) [HR06, AAC07]. As in minimum multicut, the integrality gap of (LP-spcut) is $\Omega(n^{1/7}/\operatorname{poly}\log n)$ [CK09]. Therefore, the integrality gap of (LP-spcut), and the flow-cut gap between maximum concurrent flow and sparsest cut in directed graph is polynomial in n. The non-bipartite sparsest cut problem in directed graphs is hard to approximate to within factor $2^{\Omega(\log^{1-\epsilon} n)}$ for any constant $\epsilon > 0$, assuming that some problems in **NP** do not have efficient randomized algorithms [CK09]. The bipartite sparsest cut is known to be hard to approximate to within $2^{\Omega((\log n)^{\epsilon})}$ for some $\epsilon > 0$, unless 3SAT has subexponential-time algorithms [CMM06].

4. Integral Routing

In this section we consider integral routing problems. We start with the edge-disjoint paths problem, that can be seen as the integral counterpart of maximum multicommodity flow, and discuss several closely related problems, such as node-disjoint paths and congestion minimization. We then consider an integral counterpart of the maximum concurrent flow problem, called integral concurrent flow.

Edge-disjoint paths problem (EDP) is one of the basic problems in integral routing, and we can think of it as an integral version of maximum multicommodity flow. For simplicity, in this section, we assume that all edge capacities are unit. The input to the EDP problem is an n-vertex graph $G = (V, E)$, that can be either directed or undirected, and a collection $\mathcal{M} = \{(s_1, t_1), \dots, (s_k, t_k)\}$ of k pairs of vertices, that we call *demand pairs*. In order to route a pair (s_i, t_i), we need to select a path P_i connecting s_i to t_i. The goal is to route a maximum possible number of the demand pairs via *edge-disjoint paths*: that is, every edge e may participate in at most one path in the solution. A closely related problem is node-disjoint paths (NDP), defined exactly like EDP, except that the paths chosen to route the demand pairs now need to be *node-disjoint*, so a vertex of G

may belong to at most one such path. For directed graphs, the two problems are almost equivalent: an EDP instance $(G, \mathcal{M})$ can be transformed into an instance of the NDP problem, by sub-dividing every edge of G with a new vertex. (This transformation does not preserve the number of vertices, which can grow, so if we are interested in approximation factors as a function of $|V(G)|$, we should be careful when using this transformation). An instance G of NDP in a directed graph can be transformed into an instance of the EDP problem, by replacing every vertex v with a directed edge (a_v, b_v), and every edge (u, v) with an edge (b_u, a_v). For undirected graphs, it is only known that NDP is more general than EDP, as every instance of EDP can be transformed into an instance of NDP via the same transformation, but the transformation in the opposite direction is not known for undirected graphs.

In directed graphs, both NDP and EDP are NP-hard even when the number of the demand pairs is 2 [FHW80]. The following simple algorithm achieves an $O(\sqrt{m})$-approximation for EDP, where m is the number of the graph edges [KS04, GKR+99, Kle96, KS06]. Start with the empty solution. While at least one demand pair can be routed in G, select a shortest path P, connecting any demand pair (s_i, t_i). Add P to the solution, delete all edges of P from the graph, and delete (s_i, t_i) from the list of the demand pairs that need to be routed. In order to see that this algorithm obtains an $O(\sqrt{m})$-approximation, consider any optimal solution OPT to the problem. In every iteration, if a path P connecting s_i to t_i is added to the solution, then we delete from OPT all paths sharing edges with P, and the path routing the demand pair (s_i, t_i), if such belongs to OPT. As long as P contains fewer than $\sqrt{m}$ edges, we delete at most $\sqrt{m} + 1$ paths from OPT in every iteration, while adding at least one path to our solution. Consider now the first iteration where the length of the selected path P is more than $\sqrt{m}$. Since we choose the shortest path routing any demand pair, and all paths that currently belong to OPT can be chosen by the algorithm, every path in OPT contains at least $\sqrt{m}$ edges, and, since these paths are edge-disjoint, OPT contains at most $\sqrt{m}$ paths in total. Therefore, even if we delete all paths from OPT in the current iteration, while adding only one path to the solution, we still preserve the $O(\sqrt{m})$-ratio between the number of paths deleted from OPT and the number of paths added to the solution, thus obtaining an $O(\sqrt{m})$-approximation. Surprisingly, this simple algorithm is almost the best we can hope for: EDP in directed graphs is hard to approximate to within a factor of $\Omega(m^{1/2-\epsilon})$ for any constant ϵ [GKR+99]. For the NDP problem, the algorithm described above gives an $O(\sqrt{n})$-approximation, and the problem is hard to approximate in directed graphs to within a factor of $\Omega(n^{1/2-\epsilon})$ for any constant ϵ [GKR+99].

While the approximation status of EDP and NDP is well understood in directed graphs, both problems remain wide open in undirected graphs. When the number k of the demand pairs is bounded by a constant, there is an efficient algorithm to solve both NDP and EDP [RS95, RS90a]. We discuss this algorithm in more detail in the following section. For general values of k, it is NP-hard to even decide whether all pairs can be simultaneously routed on edge-disjoint paths [Kar72]. The best currently known approximation algorithms achieve an

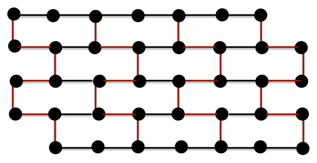
$O(\sqrt{n})$-approximation for both problems [KS04, CKS06], while the best current negative result shows that neither problem has an $O(\log^{1/2-\epsilon} n)$-approximation for any constant ϵ, unless all problems in **NP** have randomized algorithms with running time $n^{O(\text{poly}\log n)}$ [AZ05, ACG+10].

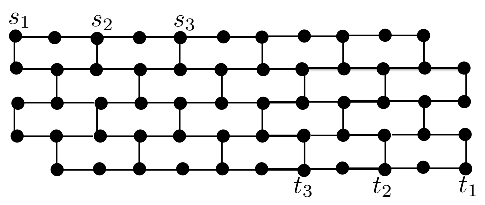
The vertices in the set $T = \{s_1, \ldots, s_k, t_1, \ldots, t_k\}$ are called *terminals*. We will assume for simplicity that all terminals are distinct, that is, $|T| = 2k$, and that each terminal is incident on exactly one edge. This can be assumed without loss of generality, by performing the simple transformation of the input graph G, that preserves the routing solutions: for each terminal $v \in T$, if v participates in z demand pairs, we add z new vertices $v_1, \ldots, v_z$ to G, each of which connects to v with an edge. We then replace v with the vertices $v_1, \ldots, v_z$ in all demand pairs in which v participates, so that each of these new vertices participates in exactly one demand pair. If we add the following integrality constraints to the linear program (LP-multi-flow):

$$f(P) \in \{0,1\} \qquad \forall 1 \le i \le k, \forall P \in \mathcal{P}_i,$$

and set the values $c(e)$ for all edges $e \in E$ in the linear program to 1, then we obtain an integral linear program, which is equivalent to EDP. Therefore, (LP-multi-flow) is an LP-relaxation of EDP. The best currently known approximation algorithm for EDP achieves an $O(\sqrt{n})$-approximation by rounding (LP-multi-flow) [CKS06]. Unfortunately, the integrality gap of (LP-multi-flow) is very large even in undirected graphs: if n denotes the number of the graph vertices, and k is the number of the demand pairs, then the integrality gap can be as large as $\Omega(\sqrt{n})$, and as large as $\Omega(k)$ [GVY93]. An example of an instance realizing this integrality gap is a wall graph. Wall graphs play an important role in algorithms for routing problems and in Graph Minor Theory. They are also among the simplest examples of graphs for which we do not have good approximation algorithms for the EDP problem. A wall of height 5 and width 4 is shown in Figure 1(a). A wall of height h and width w can be constructed from a 2-dimensional grid of width $2w$ and height h. Let $C_1, \ldots, C_{2w}$ be the columns of the grid, in their natural left-to-right order. Consider some column C_i, and let $e^i_1, \ldots, e^i_{h-1}$ be the edges of column C_i in their top-to-bottom order. If i is odd, then we delete all edges e^i_j where j is even, and if i is even, we delete all edges e^i_j where j is odd. We also delete all vertices of degree at most 1 in the resulting graph, obtaining a wall of height h and width w. This wall contains h horizontal lines corresponding to the h rows of the grid, that we call the rows of the wall, denoting them by $R_1, \ldots, R_h$ in their natural top-to-bottom order. There are also exactly w disjoint paths connecting the vertices of R_1 to the vertices of R_h, which do not contain the vertices of $R_1 \cup R_h$ as their inner vertices. We call these paths the columns of the wall, and we denote them by $C_1, \ldots, C_w$ in their natural left-to-right order.

In order to define an instance of the edge-disjoint-paths problem, we start with a wall of height $k+2$ and width $2k$. For each $1 \le i \le k$, let s_i be the unique vertex in the intersection of R_1 and C_i, and let t_i be the unique vertex in the intersection of R_{k+2} and C_{2k-i+1}. The set of the demand pairs is $\mathcal{M} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$. It is easy to see that the value of the optimal fractional solution for this instance is at least $k/2$: for each $1 \leq i \leq k$, we will define a path P_i connecting s_i to t_i, and we will send 1/2 flow unit along each such path. We will ensure that every edge of the wall belongs to at most two such paths, obtaining a feasible solution of value $k/2$ to (LP-multi-flow). In order to define path P_i, for $1 \leq i \leq k$, we start from s_i, and follow column C_i, until we reach row R_{i+1}; we then follow R_{i+1} to column C_{2k-i+1}, and column C_{2k-i+1} until we reach t_i. It is immediate to verify that every edge belongs to at most two such paths, and so setting $f(P_i) = 1/2$ for each $1 \leq i \leq k$ gives a feasible solution of value $k/2$ to (LP-multiflow). However, the value of the optimal integral solution is at most 1: assume for contradiction that we can route two demand pairs: (s_i, t_i) and (s_j, t_j), for $i \neq j$, and let P_i, P_j be the two corresponding paths. Let Γ be the cycle that serves as the boundary of the wall. The wall is a planar graph, and has a drawing in the plane, with Γ being the boundary of the outer face - this is the natural drawing, as the one in Figure 1(a). The resulting drawings of the paths P_i, P_j have to cross, since their endpoints appear in the circular order (s_i, s_j, t_i, t_j) along Γ. But this is impossible since P_i, P_j are disjoint, and the drawing is planar. Therefore, the integrality gap of (LP-multi-flow) is at least $k/2$, and, since the number of the vertices in our graph is $O(k^2)$, the gap is $\Omega(\sqrt{n})$ as a function of n.


(a) A wall of height 5 and width 4. The columns are shown in red.



(b) An integrality gap example with $k = 3$.

Figure 1. A wall graph

Interestingly, even though there are several approximation algorithms achieving constant or polylogarithmic approximation factors for large families of planar graphs, such as grids and grid-like graphs [AR95, AGLR94, KT98, KT95], The best currently known approximation ratio for EDP on planar graphs is no better than that for general graphs, namely $O(\sqrt{n})$. Even if the underlying graph is a wall of height $\Theta(\sqrt{n})$ (but the terminals can be located anywhere in the wall and not necessarily on the boundary), no better than $O(\sqrt{n})$-approximation is known, to the best of our knowledge. Closing the gap in our understanding of the approximability of EDP is one of the central problems in the area of graph routing, and a good starting point may be planar graphs or even wall graphs.

The situation with the NDP problem in undirected graphs is very similar: the best current upper and lower bounds on its approximability stand on $O(\sqrt{n})$ and $\Omega(\log^{1/2-\epsilon} n)$ for any constant ϵ, respectively [KS04, GKR$^+$99, Kle96, KS06, AZ05, ACG$^+$10]. We can again use the multi-commodity flow relaxation of the NDP problem, defined similarly to (LP-multi-flow), except that the capacity constraints are on the vertices and not on the edges of G. This relaxation has an integrality gap of $\Omega(\sqrt{n})$, and the graph realizing this gap is a 2-dimensional $(\sqrt{n} \times \sqrt{n})$-grid. No better than $O(\sqrt{n})$-approximation is known for NDP on planar graphs, and even on grid graphs.

Another important class of graphs is expander graphs. We say that a graph $G = (V, E)$ is an α-expander, iff for any subset $S \subseteq V$ of its vertices with $|S| \leq |V|/2$, $|E(S, V \setminus S)| \geq \alpha \cdot |S|$. In general, we say that a graph is an expander if it is an α-expander for some fixed constant α independent of the graph size. Both EDP and NDP have polylogarithmic approximation algorithms on bounded-degree expander graphs [LR99, BFU94, BFSU94, KR96, Fri00]. Both these problems also have constant-factor approximation algorithms on trees [GVY93, CMS07], and EDP has constant-factor approximation algorithms on grids and grid-like graphs [AR95, AGLR94, KT98, KT95].

Routing with Small Congestion Seeing that the status of the EDP problem in undirected graphs is still wide open, it is natural to investigate what happens if we relax the problem requirements slightly, by allowing small *congestion*. We say that a set of paths $\mathcal{P}$ causes edge-congestion c, if every edge belongs to at most c paths in $\mathcal{P}$. When the congestion $c = 1$, we sometimes say that $\mathcal{P}$ causes no congestion. Vertex congestion is defined similarly. It is a common practice to compare a solution to this relaxed version of EDP with an optimal solution that has no congestion. We say that an algorithm achieves an approximation factor α with congestion c for the EDP problem, iff it routes OPT/α demand pairs with congestion c, where OPT is the maximum number of pairs that can be routed with no congestion.

The classical algorithm of Raghavan and Thompson [RT87] gives a constant factor approximation for EDP with congestion $O(\log n/\log\log n)$. The algorithm performs LP-rounding of (LP-multi-flow), by viewing the values $f(P)$ for each path P as probabilities. Each path $P \in \bigcup_i \mathcal{P}_i$ is selected to the solution independently with probability $f(P)$. If several paths routing the same demand pair are selected, we discard the additional paths arbitrarily. It is not hard to show that with a constant probability we obtain a solution where the number of the demand pairs routed is within a constant of the optimal fractional solution, and each edge participates in at most $O(\log n/\log\log n)$ paths. This randomized rounding scheme can be slightly altered to give, for any congestion value c, a factor $O(cn^{1/(c-1)})$-approximation [AR06, BS00, KS04, Sri97]. More recent result give LP-rounding algorithms for EDP that achieve $O(\text{poly}\log k)$-approximation with smaller congestion [And10, Chu12, CL12], with the best current algorithm giving $O(\text{poly}\log k)$-approximation with congestion 2.

An important class of instances of the EDP problem is well-linked instances. We say that a set T of vertices in graph G is well-linked if for any pair $T_1, T_2 \subseteq T$ of equal-sized subsets of T, there is a set of $|T_1|$ node-disjoint paths connecting the vertices of T_1 to the vertices of T_2 in G. We say that an instance of EDP is well-linked if every terminal participates in exactly one demand pair, and the set of all terminals is well-linked. Chekuri, Khanna and Shepherd [CKS13, CKS05] have shown an efficient algorithm, that, given any EDP instance $(G, \mathcal{M})$, partitions it into a number of sub-instances $(G_1, \mathcal{M}_1), \ldots, (G_\ell, \mathcal{M}_\ell)$, such that each instance G_i is well-linked, while the sum of the values of the optimal fractional solutions in all these instances is $\Omega(\mathsf{OPT}/\log^2 k)$. Therefore, in order to obtain a polylogarithmic approximation with congestion 2 to EDP, it is enough to find a polylogarithmic approximation with congestion 2 in each such sub-instance separately. The main result of [CL12] is a structural theorem, that shows that any well-linked instance with k demand pairs contains a large crossbar. The crossbar can be viewed as a degree-3 tree T on $\operatorname{poly}\log k$ vertices, such that every vertex v of T is mapped to a connected subgraph C_v of G, and every edge $e = (u, v)$ of T is mapped to a collection $\mathcal{P}_e$ of $k/\operatorname{poly}\log k$ disjoint paths in G, where each path connects a vertex of C_v to a vertex of C_u. Moreover, each edge of G participates either in at most one graph in $\{C_v\}_{v \in V(T)}$, or in at most one path of $\bigcup_{e \in E(T)} \mathcal{P}_e$, but not both. This crossbar is then exploited to embed an expander X on $k/\operatorname{poly}\log k$ vertices into G with congestion at most 2. Specifically, we select a subset $\mathcal{M}' \subseteq \mathcal{M}$ of $k/\operatorname{poly}\log k$ demand pairs that we will attempt to route. Every vertex v of the expander X is mapped to a connected sub-graph H_v of G, and every edge $e = (u, v)$ of X is mapped to a path P_e in G connecting a vertex of H_v to a vertex of H_u. Each edge of G may participate in up to two sub-graphs H_v, or at most one such sub-graph and at most one path P_e. Each terminal participating in the pairs in $\mathcal{M}'$ belongs a distinct sub-graph H_{v_t} for some $v_t \in V(X)$. The embedding of the expander is performed using the crossbar, building on a beautiful result of Khandekar, Rao and Vazirani [KRV09] on constructing expanders via cut-matching games. Finally, known algorithms for routing on expander graphs are used to find the final routing.

These results demonstrate a fundamental difference between routing with congestion 1 and routing with congestion 2 or higher: Suppose we are given a solution $\mathcal{P}$ to the EDP problem that connects D of the demand pairs with congestion c, and we are interested in obtaining another solution with a lower congestion. By sending $1/c$ flow units along each path in $\mathcal{P}$, we obtain a valid fractional solution to (LP-multi-flow) of value D/c. We can then use the LP-rounding algorithm of [CL12] to find a solution connecting $\Omega\left(D/(c \operatorname{poly}\log k)\right)$ of the demand pairs with congestion 2. That is, we can lower the congestion to 2 with only a factor $(c \operatorname{poly}\log k)$ loss in the number of the demand pairs routed. However, if we are interested in routing with no congestion, then we may have to lose an $\Omega(\sqrt{n})$-factor in the number of pairs routed, as we can see from the integrality gap example described above: we can view the fractional solution as routing k demand pairs integrally with congestion 2 (by sending 1 flow unit along each path instead of $\frac{1}{2}$), but if we require an integral routing with congestion 1, then at most one pair can be routed.

The $O(\operatorname{poly}\log k)$-approximation algorithm with congestion 2 is close to the best one can hope to obtain from rounding (LP-multi-flow): as discussed above, any sub-polynomial approximation for EDP obtained via this relaxation must incur congestion at least 2. The integrality gap of (LP-multi-flow) is $\Omega\left(\left(\frac{\log n}{(\log\log n)^2}\right)^{1/(c+1)}\right)$ for any constant congestion value c [ACG$^+$10], and so the integrality gap for congestion 2 is polylogarithmic. An almost matching hardness of approximation result shows that for any constant ϵ, for any congestion value $c : 1 \leq c \leq O\left(\frac{\log\log n}{\log\log\log n}\right)$, there is no $O\left((\log n)^{\frac{1-\epsilon}{c+1}}\right)$-approximation algorithm for EDP with congestion c, unless all problems in **NP** have randomized algorithms with running time $(n^{\text{poly}\log n})$ [ACG$^+$10]. This gives an $\Omega\left(\log^{(1-\epsilon)/3} n\right)$-hardness of approximation for EDP with congestion 2. These algorithms for EDP were generalized to NDP, giving an $O(\text{poly}\log(k))$-approximation with a constant congestion [CE13].

Allowing congestion does not seem to help much in directed graphs: EDP remains $n^{\Omega(1/c)}$-hard to approximate even when congestion c is allowed, for any value c between 2 and $\delta \log n/\log\log n$, for some fixed constant δ, unless all problems in **NP** have randomized algorithms with running time $n^{\text{poly}\log n}$ [AZ06, CGKT07], almost matching the $O(cn^{1/(c-1)})$-approximation [AR06, BS00, KS04, Sri97] achievable via the randomized rounding technique.

Congestion Minimization Congestion minimization is a natural counterpart of the EDP problem: here, the goal is to route **all** demand pairs, while minimizing the edge congestion. We can slightly alter (LP-multi-flow) to obtain an LP-relaxation for the congestion minimization problem:

$$\begin{array}{llll}
\text{(LP-cong-min)} & \min & c & \\
& \text{s.t.} & & \\
& & \sum_{P\in\mathcal{P}_i} f(P) = 1 & \forall 1\leq i\leq k \\
& & \sum_{P:e\in P} f(P) \leq c & \forall e\in E \\
& & f(P)\geq 0 & \forall 1\leq i\leq k \quad \forall P\in\mathcal{P}_i
\end{array}$$

The randomized rounding algorithm of Raghavan and Thompson [RT87] gives the best currently known approximation algorithm for the congestion minimization problem, whose approximation factor is $O(\log n/\log\log n)$, by independently choosing, for each $1 \leq i \leq k$, one path $P \in \mathcal{P}_i$, where path P is chosen with probability $f(P)$. For directed graphs, this algorithm is close to being the best possible, as the problem is known to be hard to approximate to within factor $\Omega(\log n/\log\log n)$ [AZ06, CGKT07]. But for undirected graphs the problem is still wide open, with the best current negative result standing on $\Omega\left(\frac{\log\log n}{\log\log\log n}\right)$-hardness of approximation, unless all problems in **NP** have randomized algorithms with running time $(n^{\text{poly}\log n})$ [AZ07]. Even the integrality gap of (LP-cong-min) for undirected graphs is not well understood: the current upper bound stands on $O(\log n/\log\log n)$, by the algorithm of [RT87], and the current lower bound is $\Omega\left(\frac{\log\log n}{\log\log\log n}\right)$ [AZ07].

Integral Concurrent Flow In the integral concurrent flow problem (ICF), we are given an undirected n-vertex graph $G = (V, E)$, a collection $\{(s_1, t_1), \ldots, (s_k, t_k)\}$ of pairs of vertices that we call demand pairs, and a demand value D_i for each $1 \leq i \leq k$. The goal is to find a maximum value λ, and a collection $\mathcal{P}$ of paths, such that for each demand pair (s_i, t_i) set $\mathcal{P}$ contains at least $\lfloor \lambda \cdot D_i \rfloor$ paths connecting s_i to t_i, and each edge participates in at most one such path. This problem is an integral counterpart of the maximum concurrent flow problem. To the best of our knowledge, no approximation algorithms are known for the problem. As with the EDP problem, we also consider a relaxed version, where a small congestion is allowed on the edges. Chalermsook et al. [CCEL12] showed a $\operatorname{poly}\log n$-approximation algorithm for ICF with a constant congestion, by rounding solutions of an LP-relaxation similar to (LP-concurrent-flow). They also showed that for any values η, α, such that $\eta \cdot \alpha \leq O(\log\log n / \log\log\log n)$, no efficient algorithm can find an α-approximate solution with congestion η to ICF unless all problems in **NP** have randomized algorithms with running time $n^{\operatorname{poly}\log n}$.

Chalermsook at al. [CCEL12] also consider a more general version of the ICF, called group-ICF, in which, instead of the k pairs of vertices $\{(s_1, t_1), \ldots, (s_k, t_k)\}$, we are given k pairs of vertex subsets, $((S_1, T_1), \ldots, (S_k, T_k))$, so for each $1 \leq i \leq k$, $S_i, T_i \subseteq V$. The goal is to find a maximum value λ, and a collection $\mathcal{P}$ of paths, such that for each $1 \leq i \leq k$, there are at least $\lfloor \lambda \cdot D_i \rfloor$ paths connecting the vertices of S_i to the vertices of T_i in $\mathcal{P}$, and every edge $e \in E$ belongs to at most one such path. It is easy to see that group-ICF generalizes both the ICF and the EDP problems. We can use an LP-relaxation similar to (LP-concurrent-flow) for the group-ICF problem. When no congestion is allowed, the integrality gap of the relaxation is $\Omega(\sqrt{n})$, even when $k = 2$. Moreover, even if we allow congestion c, this ratio can still be as large as $\Omega(n^{1/c+1})$. Chalermsook et al. [CCEL12] show that for any $0 < \eta \leq O(\log\log n)$ and $\alpha = O\left(n^{1/2^{2\eta+3}}\right)$, no efficient algorithm can find α-approximate solutions with congestion η for group-ICF, unless all problems in **NP** have algorithms with running time $n^{O(\log\log n)}$. Given an optimal integral solution $\mathcal{P}$ to the group-ICF problem instance, let $D = \min_i \{\lfloor \lambda^* \cdot D_i \rfloor\}$ be the minimum number of paths connecting any pair (S_i, T_i) in this solution. Their hardness result only holds for the regime where $D << k$. They further show that if $D > k \operatorname{poly}\log n$, then there is an efficient algorithm that finds a $(\operatorname{poly}\log n)$-approximate solution to group-ICF with constant congestion.

5. Routing with Few Demand Pairs

In this section we consider the NDP problem on undirected graphs when the number k of the demand pairs is bounded by a constant independent of the graph size. (Recall that for directed graphs, NDP is NP-hard even for $k = 2$ [FHW80]).

Given a graph G, a *separation* of G is a pair (X, Y) of sub-graphs of G, with $X \cup Y = G$ and $E(X) \cap E(Y) = \emptyset$. The *order* of the separation is $|V(X) \cap V(Y)|$. When the number of the demand pairs is $k = 2$, the following beautiful theorem can be used to solve the NDP problem.

Theorem 5.1. *[Jun70, RS90b, Sey06, Shi80, Tho80] Let G be a graph and s_1, t_1, s_2, t_2 four vertices. Assume that there is no separation (X, Y) in G of order at most 3, such that $s_1, t_1, s_2, t_2 \in V(X)$ and $X \neq G$. Then either both pairs*

(s_1,t_1) and (s_2,t_2) can be routed on disjoint paths in G, or there is a drawing of G inside a disc in the plane, with s_1,s_2,t_1,t_2 appearing on the boundary of the disc in this circular order. Moreover, there is an efficient algorithm that either finds the routing or the drawing of G.

In order to apply the above theorem to the NDP problem instance, we pre-process the input graph G as follows: if there is a separation (X,Y) of G of order at most 3, such that $s_1,t_1,s_2,t_2 \in V(X)$ and $X \neq G$, then there must be a separation (X',Y') of G with all the above properties, such that Y' is connected. We delete from G all vertices of $V(Y')\setminus V(X')$, and add all edges connecting every pair of vertices in $V(X')\cap V(Y')$. It is easy to see that the two pairs (s_1,t_1), (s_2,t_2) can be routed on disjoint paths in the new graph iff they can be routed on disjoint paths in the old graph. We repeat this process, until G contains no separation (X,Y) of order at most 3, with $s_1,t_1,s_2,t_2 \in V(X)$ and $X \neq G$, and then apply Theorem 5.1 to find the routing.

For the case where $k>2$, but is still bounded by a constant, Robertson and Seymour [RS95, RS90a] have shown an efficient algorithm for NDP, with running time $O(n^3\cdot f(k))$, where n is the number of graph vertices, and f is some function. This running time was later improved to $O(n^2\cdot f(k))$ [KKR12]. Before we describe their algorithm, we need to define several graph-theoretic notions.

We start with the notion of treewidth. Intuitively, treewidth measures how close our graph is to a tree: the lower the treewidth value (which is always at least 1), the "closer" our graph is to being a tree. Trees are relatively simple graphs, and many combinatorial optimization problems that are NP-hard on general graphs have efficient algorithms on trees. Many other problems have good approximation algorithms on trees, even if such algorithms are not known for general graphs. Sometimes, when the graphs that we work with are not too complex, techniques used for designing algorithms on trees may still be applicable. It would be therefore useful to have some machinery that allows us to adapt known algorithms for trees to "tree-like" graphs, and to have a formal way to measure the "closeness" of a graph to a tree. The notion of treewidth achieves both these goals: it gives a way to measure the closeness of a graph to a tree, while providing a convenient tree-like representation of the graph, that often allows us to adapt the algorithms known for trees to low-treewidth graphs.

The treewidth of a graph $G=(V,E)$ is typically defined via tree decompositions. A tree-decomposition for G consists of a tree $T=(V(T),E(T))$ and a collection of sets $\{X_v\subseteq V\}_{v\in V(T)}$ called *bags*, such that the following two properties are satisfied: (i) for each edge $(a,b)\in E$, there is some node $v\in V(T)$ with both $a,b\in X_v$ and (ii) for each vertex $a\in V$, the set of all nodes of T whose bags contain a form a non-empty (connected) subtree of T. The *width* of a given tree decomposition is $\max_{v\in V(T)}\{|X_v|-1\}$, and the treewidth of a graph G, denoted by $\mathrm{tw}(G)$, is the width of a minimum-width tree decomposition for G. There is an interesting connection between graph treewidth and well-linkedness: if w denotes the size of the largest-cardinality well-linked set of vertices in G, then $w\le tw(G)\le 4w$.

The problem of computing the treewidth of a graph is NP-hard [ACP87]. When the treewidth value k is bounded by a constant, the treewidth and the corresponding tree decomposition can be computed in time $O(n \cdot f(k))$ for some function f [RS95, Lag90, Ree92, LA91, BK91, Bod93]. Using the best currently known approximation algorithms for the vertex version of the sparsest cut problem [FHL08], one can obtain an $O(\sqrt{\log k})$-approximation algorithm for computing treewidth in general graphs, together with the corresponding tree decomposition [BGHK92].

Suppose we are given an instance $(G, \mathcal{M})$ of the NDP problem, where $|\mathcal{M}| = k$, and assume that we are given a tree decomposition T of G of width w. Then the problem can be solved in time $O(n) \cdot f(w, k)$ for some function f, via dynamic programming, as follows. Using standard methods, we can transform the tree decomposition T into another tree decomposition T', such that $|V(T')| \leq n$, the width of T' is at most $w + 2k$, every vertex of T' has degree at most 3, and there is one vertex v in T whose degree is 1, and $X_v = \{s_1, \ldots, s_k, t_1, \ldots, t_k\}$. We root the tree T at the vertex v. For each vertex u of T, let S_u be the set of all the vertices of T contained in the sub-tree rooted at u, and let $Y_u = \bigcup_{u' \in S_u} X_{u'}$. We define a graph G_u associated with the vertex u to be the sub-graph of G induced by Y_u. We will think of the vertices of X_u as the terminals for the graph G_u. Since $|X_u| \leq w + 2k + 1$, there are $2^{O(w+k)}$ ways to define a matching M between the vertices of X_u (where we allow the matchings to be partial). We say that a matching M is *routable* in G_u iff there is a solution to the NDP problem in graph G_u, where every pair of vertices in M is routed. A *folio* of the vertex $u \in V(T)$, denoted by $\pi(u)$, is the list of all such matchings M (defined over the set X_u of vertices), such that M is routable in G_u. The main idea of the algorithm is to use dynamic programming in order to compute a folio for every vertex $u \in V(T)$, by processing the tree in the bottom-up fashion. For each matching M in the folio $\pi(u)$, we will also compute and store the set of paths in G_u routing the matching M. Notice that once we compute $\pi(v)$, we can select a matching $M \in \pi(v)$ containing the largest number of the demand pairs from $\mathcal{M}$ to obtain a solution to the NDP problem. For each leaf vertex $u \in V(T)$, the folio $\pi(u)$ can be computed by exhaustively going over all possible matchings M, and for each such matching, checking whether it can be routed in G_u by exhaustive search, since $|V(G_u)| \leq w + 2k$. When a non-leaf vertex u is processed, we need to check all possible ways to combine the matchings in the folios $\pi(u')$, $\pi(u'')$ of the two children u', u'' of u into a single folio $\pi(u)$ of u. Since the sizes of all three folios are bounded by $2^{O(w+k)}$, and $|X_u| \leq 2k + w$, the running time of this algorithm can be bounded by $f(k + w)$ for some function f, and the overall running time $O(n \cdot f(k + w))$. This gives an efficient algorithm for NDP with constant number of demand pairs on bounded-treewidth graphs. But what about general graphs, whose treewidth may not be bounded by a constant? Robertson and Seymour's Excluded Grid Theorem is a very powerful tool for handling such graphs. The theorem states that there is some function $g : \mathbb{Z}^+ \to \mathbb{Z}^+$, such that for any integer t, every graph of treewidth at least $g(t)$ contains a sub-division of the $(t \times t)$-wall (this is equivalent to saying that G contains a $(t \times t)$-grid as a minor). A long line of work is dedicated to improving the known upper and lower bounds on the function g [RS86, RST94, DJGT99, Die12, KK12, LS12, CC14]. The best current bounds show that the theorem holds for $g(t) = O(t^{98} \cdot \text{poly}\log(t))$ [CC14], and the best negative result shows that $g(t) = \Omega(t^2 \log t)$ must hold [RST94]. Robertson et al. [RST94] suggest that this value may be sufficient, and Demaine et al. [DHK09] conjecture that the bound of $g(t) = \Theta(t^3)$ is both necessary and sufficient.

Notice that if the treewidth of G is w, then there is a well-linked set of size $\Omega(w)$ in G. We can then use the machinery developed for approximating EDP and NDP in well-linked instances. The first step in the proof of the excluded grid theorem of [CC14] constructs a crossbar in G, given the set of $\Omega(w)$ well-linked vertices. This step expands and generalizes the crossbar construction from [CL12, CE13]. In the next step, a new crossbar is constructed, where the underlying tree is a path, and then a result of Leaf and Seymour [LS12] is used to build a large wall in this new crossbar.

We are now ready to complete the description of the algorithm for NDP when the number k of the demand pairs is bounded by a constant. We use some threshold function $\tau(k)$. If the treewidth of graph G is at most $\tau(k)$, then we run the dynamic programming algorithm described above to solve the NDP problem in time $O(n \cdot f(\tau(k)))$. Otherwise, the treewidth of G is at least $\tau(k)$, and we can find a large wall in G. Using this wall, we can identify an *irrelevant vertex* v in G, such that for any subset $\mathcal{M}' \subseteq \mathcal{M}$ of the demand pairs, the pairs in $\mathcal{M}'$ are simultaneously routable in $G \setminus \{v\}$ iff they are simultaneously routable in G. We then delete the vertex v from graph G and continue. Since the number of iteration is bounded by $|V(G)|$, we will eventually arrive at a graph G whose treewidth is at most $\tau(k)$, and then apply the dynamic programming algorithm to it.

Acknowledgements The author thanks Chandra Chekuri and David Kim for their comments on an earlier version of this survey.

References

[AAC07] Amit Agarwal, Noga Alon, and Moses S. Charikar. Improved approximation for directed cut problems. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 671–680, New York, NY, USA, 2007. ACM.

[ACG+10] Matthew Andrews, Julia Chuzhoy, Venkatesan Guruswami, Sanjeev Khanna, Kunal Talwar, and Lisa Zhang. Inapproximability of edge-disjoint paths and low congestion routing on undirected graphs. *Combinatorica*, 30(5):485–520, 2010.

[ACP87] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[AGLR94] Baruch Awerbuch, Rainer Gawlick, Tom Leighton, and Yuval Rabani. On-line admission control and circuit routing for high performance computing and communication. In *Proc. 35th IEEE Symp. on Foundations of Computer Science*, pages 412–423, 1994.

[AGS13] Sanjeev Arora, Rong Ge, and Ali Kemal Sinop. Towards a better approximation for sparsest cut? In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 270–279. IEEE, 2013.

[ALN05] Sanjeev Arora, James R. Lee, and Assaf Naor. Euclidean distortion and the sparsest cut. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 553–562, New York, NY, USA, 2005. ACM.

[AMS07] Christoph Ambuhl, Monaldo Mastrolilli, and Ola Svensson. Inapproximability results for sparsest cut, optimal linear arrangement, and precedence constrained scheduling. In *FOCS '07: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 329–337, Washington, DC, USA, 2007. IEEE Computer Society.

[And10] Matthew Andrews. Approximation algorithms for the edge-disjoint paths problem via Raecke decompositions. In *Proceedings of IEEE FOCS*, pages 277–286, 2010.

[AR95] Yonatan Aumann and Yuval Rabani. Improved bounds for all optical routing. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, SODA '95, pages 567–576, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.

[AR98] Yonatan Aumann and Yuval Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM J. Comput.*, 27(1):291–301, 1998.

[AR06] Yossi Azar and Oded Regev. Combinatorial algorithms for the unsplittable flow problem. *Algorithmica*, 44(1):49–66, 2006.

[ARV09] Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56(2), 2009.

[AZ05] Matthew Andrews and Lisa Zhang. Hardness of the undirected edge-disjoint paths problem. In *STOC*, pages 276–283. ACM, 2005.

[AZ06] Matthew Andrews and Lisa Zhang. Logarithmic hardness of the directed congestion minimization problem. In *STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 517–526, New York, NY, USA, 2006. ACM.

[AZ07] Matthew Andrews and Lisa Zhang. Hardness of the undirected congestion minimization problem. *SIAM J. Comput.*, 37(1):112–131, 2007.

[BFSU94] Andrei Z. Broder, Alan M. Frieze, Stephen Suen, and Eli Upfal. Optimal construction of edge-disjoint paths in random graphs. In *Proc. 5th ACM-SIAM SODA*, pages 603–612, 1994.

[BFU94] Andrei Z. Broder, Alan M. Frieze, and Eli Upfal. Existence and construction of edge-disjoint paths on expander graphs. *SIAM J. Comput.*, pages 976–989, 1994.

[BGHK92] Hans L Bodlaender, John R Gilbert, Hjálmtỳr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, and minimum elimination tree height. In *Graph-Theoretic Concepts in Computer Science*, pages 1–12. Springer, 1992.

[BK91] Hans L Bodlaender and Ton Kloks. Better algorithms for the pathwidth and treewidth of graphs. In *Automata, Languages and Programming*, pages 544–555. Springer, 1991.

[Bod93] Hans L Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 226–234. ACM, 1993.

[BS00] Alok Baveja and Aravind Srinivasan. Approximation algorithms for disjoint paths and related routing and packing problems. *Mathematics of Operations Research*, 25:2000, 2000.

[CC14] Chandra Chekuri and Julia Chuzhoy. Polynomial bounds for the grid-minor theorem. In *STOC*, 2014.

[CCEL12] Parinya Chalermsook, Julia Chuzhoy, Alina Ene, and Shi Li. Approximation algorithms and hardness of integral concurrent flow. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 689–708, New York, NY, USA, 2012. ACM.

[CE13] Chandra Chekuri and Alina Ene. Poly-logarithmic approximation for maximum node disjoint paths with constant congestion. In *Proc. of ACM-SIAM SODA*, 2013.

[CGKT07] Julia Chuzhoy, Venkatesan Guruswami, Sanjeev Khanna, and Kunal Talwar. Hardness of routing with congestion in directed graphs. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 165–178, New York, NY, USA, 2007. ACM.

[Chu12] Julia Chuzhoy. Routing in undirected graphs with constant congestion. In *Proc. of ACM STOC*, pages 855–874, 2012.

[CK09] Julia Chuzhoy and Sanjeev Khanna. Polynomial flow-cut gaps and hardness of directed cut problems. *Journal of the ACM (JACM)*, 56(2):6, 2009.

[CKK$^+$06] Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *Comput. Complex.*, 15(2):94–114, 2006.

[CKR05] Joseph Cheriyan, Howard Karloff, and Yuval Rabani. Approximating directed multicuts. *Combinatorica*, 25(3):251–269, 2005.

[CKS05] Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. Multicommodity flow, well-linked terminals, and routing problems. In *Proc. of ACM STOC*, pages 183–192, 2005.

[CKS06] Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. An $O(\sqrt{n})$ approximation and integrality gap for disjoint paths and unsplittable flow. *Theory of Computing*, 2(1):137–146, 2006.

[CKS13] Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. The all-or-nothing multicommodity flow problem. *SIAM Journal on Computing*, 42(4):1467–1493, 2013.

[CL12] Julia Chuzhoy and Shi Li. A polylogarithimic approximation algorithm for edge-disjoint paths with congestion 2. In *Proc. of IEEE FOCS*, 2012.

[CMM06] Moses Charikar, Konstantin Makarychev, and Yury Makarychev. Directed metrics and directed graph partitioning problems. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 51–60, New York, NY, USA, 2006. ACM.

[CMS07] Chandra Chekuri, Marcelo Mydlarz, and F. Bruce Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Trans. Algorithms*, 3, August 2007.

[DHK09] Erik Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Algorithmic graph minor theory: Improved grid minor bounds and Wagner's contraction. *Algorithmica*, 54:142–180, 2009.

[Die12] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[DJGT99] Reinhard Diestel, Tommy R. Jensen, Konstantin Yu. Gorbunov, and Carsten Thomassen. Highly connected sets and the excluded grid theorem. *J. Comb. Theory, Ser. B*, 75(1):61–73, 1999.

[DJP$^+$94] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23(4):864–894, 1994.

[FF62] L.R. Ford and D.R. Fulkerson. Flows in networks. Princeton University Press, Princeton, NJ, 1962.

[FHL08] U. Feige, M.T. Hajiaghayi, and J.R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38:629–657, 2008.

[FHW80] Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980.

[Fri00] Alan M. Frieze. Edge-disjoint paths in expander graphs. *SIAM Journal On Computing*, 30:2001, 2000.

[GKR$^+$99] Venkatesan Guruswami, Sanjeev Khanna, Rajmohan Rajaraman, Bruce Shepherd, and Mihalis Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. In *Journal of Computer and System Sciences*, page pages, 1999.

[GS11] Venkatesan Guruswami and Ali Kemal Sinop. Lasserre hierarchy, higher eigenvalues, and approximation schemes for graph partitioning and quadratic integer programming with PSD objectives. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 482–491. IEEE, 2011.

[Gup03] Anupam Gupta. Improved results for directed multicut. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 454–455, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[GVY93] Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees, with applications to matching and set cover. In Andrzej Lingas, Rolf G. Karlsson, and Svante Carlsson, editors, *ICALP*, volume 700 of *Lecture Notes in Computer Science*, pages 64–75. Springer, 1993.

[GVY96] Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM Journal on Computing*, 25:698–707, 1996.

[HR06] Mohammad Taghi Hajiaghayi and Harald Räcke. An $O(\sqrt{n})$-approximation algorithm for directed sparsest cut. *Inf. Process. Lett.*, 97(4):156–160, 2006.

[Jun70] H. A. Jung. Eine verallgemeinerung des n-fachen zusammenhangs fr graphen. *Math. Ann.*, 187:95—103, 1970.

[Kar72] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[KARR90] Philip N Klein, Ajit Agrawal, R Ravi, and Satish Rao. Approximation through multicommodity flow. In *FOCS*, volume 31, pages 726–737, 1990.

[Kho02] Subhash Khot. On the power of unique 2-prover 1-round games. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 767–775, New York, NY, USA, 2002. ACM.

[KK12] K. Kawarabayashi and Y. Kobayashi. Linear min-max relation between the treewidth of H-minor-free graphs and its largest grid minor. In *Proc. of STACS*, 2012.

[KKR12] Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 102(2):424–435, 2012.

[Kle96] Jon Michael Kleinberg. *Approximation algorithms for disjoint paths problems.* PhD thesis, Citeseer, 1996.

[KR96] Jon Kleinberg and Ronitt Rubinfeld. Short paths in expander graphs. In *In Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 86–95, 1996.

[KRV09] Rohit Khandekar, Satish Rao, and Umesh Vazirani. Graph partitioning using single commodity flows. *J. ACM*, 56(4):19:1–19:15, July 2009.

[KS04] Stavros G. Kolliopoulos and Clifford Stein. Approximating disjoint-path problems using packing integer programs. *Mathematical Programming*, 99:63–87, 2004.

[KS06] Petr Kolman and Christian Scheideler. Improved bounds for the unsplittable flow problem. *J. Algorithms*, 61(1):20–44, 2006.

[KT95] Jon M. Kleinberg and Éva Tardos. Disjoint paths in densely embedded graphs. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 52–61, 1995.

[KT98] Jon M. Kleinberg and Éva Tardos. Approximations for the disjoint paths problem in high-diameter planar networks. *J. Comput. Syst. Sci.*, 57(1):61–73, 1998.

[KV05] Subhash A. Khot and Nisheeth K. Vishnoi. The unique games conjecture, integrality gap for cut problems and embeddability of negative type metrics into ℓ_1. In *FOCS '05: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 53–62, Washington, DC, USA, 2005. IEEE Computer Society.

[LA91] Jens Lagergren and Stefan Arnborg. Finding minimal forbidden minors using a finite congruence. In *Automata, Languages and Programming*, pages 532–543. Springer, 1991.

[Lag90] Jens Lagergren. Efficient parallel algorithms for tree-decomposition and related problems. In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on*, pages 173–182. IEEE, 1990.

[LLR95] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15:215–245, 1995.

[LR99] F. T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46:787–832, 1999.

[LS12] Alexander Leaf and Paul Seymour. Treewidth and planar minors. Manuscript, available at https://web.math.princeton.edu/ pds/papers/treewidth/paper.pdf, 2012.

[Men27] Karl Menger. Zur allgemeinen kurventheorie. *Fund. Math*, 10:96–115, 1927.

[Ree92] Bruce A Reed. Finding approximate separators and computing tree width quickly. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 221–228. ACM, 1992.

[RS86] Neil Robertson and P D Seymour. Graph minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, August 1986.

[RS90a] N. Robertson and P. D. Seymour. Outline of a disjoint paths algorithm. In *Paths, Flows and VLSI-Layout.* Springer-Verlag, 1990.

[RS90b] N. Robertson and P.D. Seymour. Graph minors. IX. Disjoint crossed paths. *J. Comb. Theory Ser. B*, 49(1):40–77, June 1990.

[RS95] Neil Robertson and Paul D Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995.

[RST94] N Robertson, P Seymour, and R Thomas. Quickly Excluding a Planar Graph. *Journal of Combinatorial Theory, Series B*, 62(2):323–348, November 1994.

[RT87] Prabhakar Raghavan and Clark D. Tompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, December 1987.

[Sey06] Paul D. Seymour. Disjoint paths in graphs. *Discrete Mathematics*, 306(10-11):979–991, 2006.

[Shi80] Yossi Shiloach. A polynomial solution to the undirected two paths problem. *J. ACM*, 27(3):445–456, 1980.

[Sri97] Aravind Srinivasan. Improved approximations for edge-disjoint paths, unsplittable flow, and related routing problems. In *IEEE Symposium on Foundations of Computer Science*, pages 416–425, 1997.

[SSZ04] Michael Saks, Alex Samorodnitsky, and Leonid Zosin. A lower bound on the integrality gap for minimum multicut in directed networks. *Combinatorica*, 24(3):525–530, 2004.

[Tho80] C. Thomassen. 2-linked graphs. *Erop. J. Combinatorics*, 1:371—378, 1980.

Toyota Technological Institute at Chicago, 6045 S. Kenwood Ave., Chicago IL 60637
E-mail: cjulia@ttic.edu