In: E. Bampis et al. (Eds.): Efficient Approximation and Online Algorithms, LNCS 3484, Springer, 2006, pp. 97–134.

Approximation Algorithms for Edge-Disjoint Paths and Unsplittable Flow

Thomas Erlebach*

Department of Computer Science, University of Leicester, UK.
t.erlebach@mcs.le.ac.uk

Abstract. In the maximum edge-disjoint paths problem (MEDP) the input consists of a graph and a set of requests (pairs of vertices), and the goal is to connect as many requests as possible along edge-disjoint paths. We give a survey of known results about the complexity and approximability of MEDP and sketch some of the main ideas that have been used to obtain approximation algorithms for the problem. We consider also the generalization of MEDP where the edges of the graph have capacities and each request has a profit and a demand, called the unsplittable flow problem.

1 Introduction

Optimization problems concerning edge-disjoint paths in graphs have led to a number of interesting approximation results in the last decade. One of the main reasons for studying such problems is that they are encountered in modern communication networks where establishing a connection requires reserving a certain amount of bandwidth on all edges along some path from the sender to the receiver. If the network does not have sufficient bandwidth to satisfy all connection requests, some requests must be rejected and it is meaningful to try to maximize the number of accepted requests. The optimization problem that seems to lie at the heart of this call admission control problem is the maximum edge-disjoint paths problem (MEDP).

An instance of MEDP is given by a directed graph $G = (V, E)$ and a set (or multiset) containing k requests $\mathcal{R} = \{(s_i, t_i) \mid i = 1, \ldots, k\}$, where each request is a pair of vertices in V. The request (s_i, t_i) asks for a directed path from s_i to t_i in G. We often use "request i" and "request (s_i, t_i)" interchangeably. A feasible solution is given by a subset $\mathcal{A}$ of $\mathcal{R}$ and an assignment of edge-disjoint paths to all requests in that subset. More precisely, each $(s_i, t_i) \in \mathcal{A}$ must be assigned a directed path π_i from s_i to t_i in G such that no two paths π_i and π_j (for $i, j \in \mathcal{A}$ and $i \neq j$) have a directed edge of the graph in common. Such a subset $\mathcal{A}$ is called *realizable* and a set of edge-disjoint paths assigned to the

* Work partly done while the author was affiliated with ETH Zurich and supported by EU Thematic Network APPOL II (IST-2001-32007), with funding provided by the Swiss Federal Office for Education and Science, and by the Swiss National Science Foundation under Contract No. 2100-63563.00 (AAPCN).

requests in $\mathcal{A}$ is called an *edge-disjoint routing* for $\mathcal{A}$. The goal is to maximize the cardinality of $\mathcal{A}$, denoted by $|\mathcal{A}|$. The requests in $\mathcal{A}$ are called the *accepted requests*, those in $\mathcal{R} \setminus \mathcal{A}$ the *rejected requests*.

MEDP can also be studied for undirected graphs. In this case, a request (s_i, t_i) asks for an undirected path connecting s_i and t_i in the given undirected graph, and two paths are edge-disjoint if they do not share an undirected edge. In any case, MEDP can be stated in a compact way as follows.

Problem MEDP (Maximum Edge-Disjoint Paths)
Input: graph $G = (V, E)$, requests $\mathcal{R} = \{(s_i, t_i) \mid i = 1, \ldots, k\}$
Feasible solution: realizable subset $\mathcal{A} \subseteq \mathcal{R}$ and edge-disjoint routing for $\mathcal{A}$
Goal: maximize $|\mathcal{A}|$

In general, the same vertex may appear several times as an endpoint of a request in $\mathcal{R}$, and we may even have several identical requests in $\mathcal{R}$. On the other hand, if a set $\mathcal{R}$ of requests has the property that every vertex of G is the endpoint of at most one request in $\mathcal{R}$, the set $\mathcal{R}$ is called a *partial permutation*.

As it turns out that MEDP is an $\mathcal{NP}$-hard optimization problem in general, one is interested in identifying special cases that can be solved optimally in polynomial time and in deriving good approximation algorithms for the other cases. An algorithm for MEDP is a ρ-approximation algorithm if it runs in polynomial time and always outputs a feasible solution $\mathcal{A}$ satisfying $|\mathcal{A}| \geq OPT/\rho$, where OPT denotes the cardinality of an optimal solution. For randomized algorithms, the value $|\mathcal{A}|$ in this definition is replaced by its expectation, taken over the random choices of the algorithm. The approximation ratio ρ can also be a function of certain parameters of the given instance, for example, of the number of vertices or edges of the given graph. We always let $n = |V|$ and $m = |E|$.

Some of the algorithms that we will encounter work also for the *on-line* version of the problem, where the requests are presented to the algorithm one by one in arbitrary order and the algorithm must accept or reject each request without knowledge of future requests. In this case, we will say that the algorithm is an *on-line algorithm*.

In this chapter, we give a tutorial survey of known results concerning approximation algorithms for MEDP. We treat results for arbitrary graphs as well as results for specific graph classes. In Section 1.1, we define all the specific graph classes that we deal with as well as some graph parameters. In Section 1.2, we give a brief survey of known complexity results for MEDP, i.e., we explain which special cases can be solved optimally in polynomial time and which variants are known to be $\mathcal{NP}$-hard or $\mathcal{APX}$-hard.

Section 2 discusses approximation algorithms for MEDP in arbitrary graphs. Sections 2.1 to 2.3 analyze various greedy-type algorithms, some of which achieve approximation ratio $O(\sqrt{m})$. In Section 2.4, we give an inapproximability result showing that no better approximation ratio (as a function of m) can be achieved for arbitrary directed graphs. Section 2.5 deals with the linear programming relaxation of MEDP and shows that although the integrality gap can be very large in general, randomized rounding yields a constant-factor approximation algorithm for a generalization of MEDP with large edge capacities.

Results for specific graph classes are reviewed in Section 3. We discuss results for trees, trees of rings, meshes, hypercubes, de Bruijn graphs, expanders, and complete graphs. In Section 4 we briefly comment on the version of MEDP with pre-determined paths. In Section 5 we consider a generalization of MEDP with edge capacities, demand values, and profit values, called the *unsplittable flow problem*. We present combinatorial algorithms for arbitrary directed graphs and give a summary of results that have been achieved using linear programming techniques. In Section 6, we mention further results that are related to MEDP and the unsplittable flow problem.

1.1 Definition of Graph Classes and Graph Parameters

We give brief definitions of the graph classes that we consider later on. First, consider undirected graphs. A graph is a *complete graph* or a *clique* if there is an edge between every pair of vertices. A *chain* is a graph that consists of a single path. A *tree* is a connected acyclic graph. A tree in which all vertices except one have degree 1 is called a *star*. A *spider* or *generalized star* is a tree in which at most one vertex can have degree larger than 2. A *ring* is a graph that consists of a single cycle. A connected graph all of whose biconnected components are rings is called a *tree of rings*.

The (two-dimensional) $n_1 \times n_2$ *mesh* is the graph with vertex set $V = \{(x, y) \in \mathbb{N}^2 \mid 1 \leq x \leq n_1, 1 \leq y \leq n_2\}$ and with vertex (x, y) being adjacent to the vertices in $\{(x-1, y), (x+1, y), (x, y+1), (x, y-1)\} \cap V$. While all internal vertices have degree 4, the vertices on the boundary of the mesh have only two or three neighbors.

The *hypercube* of dimension d is the graph with 2^d vertices corresponding to the 2^d binary strings of length d, and with an edge between two vertices if and only if the respective binary strings differ only in one position. There are several constant-degree graphs that are called *hypercubic networks*: The d-dimensional *butterfly* is a graph whose vertices are pairs (w, i), where i is an integer satisfying $0 \leq i \leq d$ and w is a binary string of length d. For a vertex (w, i), we call i its *layer*. There is an edge between two vertices (w, i) and (u, j) if and only if $j = i+1$ and either $w = u$ or w and u differ precisely in the jth bit from the left. The binary d-dimensional *de Bruijn* graph is a directed graph with 2^d vertices corresponding to the 2^d binary strings of length d. It has a directed edge from u to v if and only if the binary representation of v can be obtained from the binary representation of u by a cyclic left-shift or by a cyclic left-shift followed by flipping the rightmost bit. An undirected de Bruijn graph is obtained from a directed de Bruijn graph by ignoring the directions of the edges and removing self-loops.

A *bidirected* graph is a graph that is obtained from an undirected graph by replacing each undirected edge with two directed edges of opposite directions. For any of the graph classes defined above, we can thus also consider its bidirected counterpart.

The maximum degree of a graph G is denoted by $\Delta(G)$. For a given graph $G = (V, E)$ and a set $X \subseteq V$, we denote by $\delta(X)$ the set of edges with one

endpoint in X and the other endpoint in $V \setminus X$. The *expansion* (or *flux*) $\beta(G)$ of the graph G is defined as

$$\beta(G) = \min_{X \subset V, |X| \leq \frac{|V|}{2}} \frac{|\delta(X)|}{|X|}.$$

A graph is called an *expander* if its expansion is bounded from below by some constant.

If the graph G is clear from the context, we write Δ and β for $\Delta(G)$ and $\beta(G)$, respectively.

The *treewidth* of a graph with n nodes is a value between 1 and $n-1$ that measures (in some sense) how "tree-like" a graph is. For example, a graph has treewidth 1 if and only if all its connected components are trees. We do not give the definition of treewidth here and refer the reader to [7, 8].

1.2 Polynomial-Time Solvable Cases and Hardness Results

Most of the early work on edge-disjoint paths problems has focused on the version of the problem where the goal is to either route *all* given requests along edge-disjoint paths or certify that such a routing does not exist. This decision problem is one of the classical $\mathcal{NP}$-complete problems [27]. For directed graphs, Fortune, Hopcroft, and Wyllie [23] proved that the problem is $\mathcal{NP}$-complete even if only two terminal pairs are given. The results in the graph minor series by Robertson and Seymour led to a polynomial algorithm for undirected graphs and any constant number of terminal pairs [51]. If the number of terminal pairs is arbitrary, the problem was shown $\mathcal{NP}$-complete for meshes in [42] and in planar graphs of maximum degree 3 by Middendorf and Pfeiffer [46]. Substantial effort has been devoted to the identification of polynomial-time solvable special cases; we refer the reader to the surveys by Frank [24] and Vygen [57].

An intensive investigation of the maximization version MEDP started in the 1990s and is still continuing. Observe that there are classes of graphs for which it can be checked in polynomial time whether a set $\mathcal{R}$ of requests is realizable, but if the answer is *no*, it is $\mathcal{NP}$-hard to compute a maximum subset of realizable requests. Bidirected trees and undirected or bidirected trees of rings are examples.

Nevertheless, MEDP can be solved optimally in polynomial time for some classes of graphs. A first such class are chains. The routing for each request in a chain is uniquely determined by its endpoints (the same is true for arbitrary trees). Therefore, a set of requests in a chain can be represented as a set of intervals on the real line, and it suffices to find a maximum number of pairwise disjoint intervals. This problem has been studied in the context of interval graphs and is known to be solvable in linear time [30]. Thus, MEDP can be solved optimally in polynomial time for undirected or bidirected chains.

For undirected trees, a polynomial algorithm for MEDP has been found by Garg, Vazirani and Yannakakis [28].

For bidirected trees of arbitrary degree, MEDP has been proved $\mathcal{APX}$-complete by Erlebach and Jansen [21], implying that there is a constant $r > 1$ such that no polynomial-time algorithm can achieve approximation ratio r unless $\mathcal{P} = \mathcal{NP}$. The proof is based on the proof by Garg, Vazirani, and Yannakakis [28] showing that a generalization of MEDP with edge capacities is $\mathcal{APX}$-complete in undirected trees with edge capacities in $\{1, 2\}$. It can also be adapted to trees of rings, showing that MEDP is $\mathcal{APX}$-complete for undirected and bidirected trees of rings [20].

In bidirected stars, each request uses at most two edges, and the MEDP problem can be reduced to the maximum bipartite matching problem. This approach can be extended to give a polynomial-time algorithm for bidirected spiders as well, as shown for a more general problem by Erlebach and Vukadinović [22]. In undirected or bidirected rings, MEDP can also be solved optimally in polynomial time [58, 49].

It is known that many hard problems can be solved efficiently on graphs of bounded treewidth [7]. However, since trees of rings have treewidth 2, MEDP is $\mathcal{APX}$-hard even for graphs with treewidth 2. On the other hand, it is remarked in [19] that MEDP can be solved optimally in polynomial time on graphs whose treewidth and maximum degree are both bounded by a constant. Furthermore, Zhou et al. [59] have shown that MEDP can be solved optimally in polynomial time on graphs of bounded treewidth if the number of requests is $O(\log n)$ or if certain conditions on the location of the request endpoints are fulfilled (including the case when the union of the graph and the demand graph has bounded treewidth, where the demand graph is the graph with an edge between the endpoints of each request).

It is interesting to note that for undirected graphs, no stronger inapproximability result than $\mathcal{APX}$-hardness is known for MEDP. For directed graphs, we will see a strong inapproximability result in Section 2.4.

2 Edge-Disjoint Paths in Arbitrary Graphs

2.1 The Greedy Algorithm

A very natural algorithm for MEDP is the greedy algorithm (see Fig. 1). It processes the requests in arbitrary order. For each request, it checks whether the request can be routed along a path that is edge-disjoint from all paths assigned to previously accepted requests. If so, it accepts the request and routes it along some shortest path that does not intersect previously accepted paths.

Unfortunately, this algorithm does not achieve a good approximation ratio in general. For example, consider a chain of n vertices, numbered from 1 to n. Assume that the first request processed by the algorithm is $(1, n)$ and that there are $n-1$ further requests of the form $(i, i+1)$, for $i = 1, 2, \ldots, n-1$. The greedy algorithm accepts the first request but no other request, because the first request blocks all $n - 1$ edges of the chain. The optimal solution, however, consists of the other $n - 1$ requests. Thus, the optimal solution is better than the greedy solution by a factor of $n - 1$.

```
algorithm Greedy(G = (V, E), R = {(s_i, t_i) | i = 1, ..., k}):
  A ← ∅;
  for i = 1 to k do
      if ∃ path from s_i to t_i in G then
          A ← A ∪ {(s_i, t_i)};
          π_i ← a shortest path from s_i to t_i in G;
          remove all edges of π_i from G;
      fi;
  od;
  return A and {π_i | (s_i, t_i) ∈ A};
```

Fig. 1. The greedy algorithm.

On the other hand, we can show that the optimal solution can never be better than the greedy solution by more than a factor of $n-1$. To see this, we employ the following proof technique. Let $\mathcal{A}^*$ be the set of requests in an arbitrary optimal solution and let π_i^* be the path assigned to request $(s_i, t_i) \in \mathcal{A}^*$ by this solution. Let $\mathcal{A}$ and π_i be defined analogously for the greedy solution. We consider the execution of the greedy algorithm and, whenever it accepts a request i, we remove from $\mathcal{A}^*$ the request i (provided $i \in \mathcal{A}^*$) and all other requests in $\mathcal{A}^*$ whose optimal path π_j^* intersects the greedy path π_i. Note that at any time of the execution, the paths π_j^* of the requests that remain in $\mathcal{A}^*$ are disjoint from the paths of the requests that have already been accepted by the greedy algorithm. When the greedy algorithm terminates, the set $\mathcal{A}^*$ must be empty, by definition of the greedy algorithm. Therefore, if we can show that for each request accepted by the greedy algorithm, we have to remove at most ρ paths from the optimal solution, this implies that $|\mathcal{A}^*| \leq \rho \cdot |\mathcal{A}|$ and, consequently, the greedy algorithm achieves approximation ratio at most ρ.

So how many requests do we have to remove from $\mathcal{A}^*$ when the greedy algorithm accepts a request i? In a graph with n vertices, every simple path (and therefore also π_i) consists of at most $n-1$ edges. Consequently, it can intersect at most $n-1$ paths π_j^* of requests in $\mathcal{A}^*$. Since it suffices to remove these up to $n-1$ paths as well as the request i itself, this shows that we have to remove at most n paths from $\mathcal{A}^*$. In fact, we can strengthen this to show that we have to remove at most $n-1$ paths: If the path π_i consists of fewer than $n-1$ edges, the above argument can be used. If π_i has length $n-1$, it does in fact pass through all vertices of the graph. We would have to remove n paths from $\mathcal{A}^*$ only if π_i^* was edge-disjoint from π_i, but this cannot happen since π_i passes through all vertices and is a shortest path from s_i to t_i (among all paths that do not intersect previously accepted paths). Thus, the approximation ratio of the greedy algorithm is bounded from above by $n-1$ as well.

Theorem 1. *The greedy algorithm has approximation ratio $n-1$ for* MEDP *in directed or undirected graphs with n vertices, and this bound is tight.*

```
algorithm BoundedGreedy(G = (V, E), R = {(s_i, t_i) | i = 1, ..., k}, D):
  A ← ∅;
  for i = 1 to k do
      if ∃ path of length at most D from s_i to t_i in G then          (*)
          A ← A ∪ {(s_i, t_i)};
          π_i ← a shortest path from s_i to t_i in G;
          remove all edges of π_i from G;
      fi;
  od;
  return A and {π_i | (s_i, t_i) ∈ A};
```

Fig. 2. The bounded-length greedy algorithm. It differs from the standard greedy algorithm only by the modified condition in line (*).

2.2 The Bounded-Length Greedy Algorithm

The greedy algorithm can perform badly because it may accept a request and route it on a very long path while the optimal solution accepts many requests whose paths intersect the long path instead. One idea to avoid this problem is to restrict the length of the paths that the greedy algorithm is allowed to use. The bounded-length greedy algorithm, suggested by Kleinberg [33] and shown in Fig. 2, takes an additional parameter D and behaves like the greedy algorithm, but accepts a request only if it can be routed on a path of length at most D. Thus, each accepted request can intersect the paths of at most D other requests in the optimal solution. On the other hand, the algorithm rejects all requests whose endpoints are at distance larger than D. This means that the algorithm will have approximation ratio ∞ on instances where all requests have endpoints at distance larger than D. Therefore, we need to check for this case separately and arrive at the following algorithm, which we call *BoundedGreedy*$'$.

1. If for all requests (s_i, t_i) the length of a shortest path from s_i to t_i is at least $D + 1$, return the solution consisting of a single request (s_i, t_i) and an arbitrary path from s_i to t_i assigned to it.
2. Otherwise, run *BoundedGreedy*$(G, \mathcal{R}, D)$.

To analyze this algorithm, let us first consider the case that the condition of step 1 applies and that the algorithm accepts a single request. Since all paths assigned to a request must use at least $D + 1$ of the m edges of the graph, even the optimal solution can contain at most $m/(D+1)$ paths. Thus, the ratio is at most $m/(D + 1)$ in this case. Now assume that the algorithm *BoundedGreedy* is actually called. Each request accepted by *BoundedGreedy* (and thus also by *BoundedGreedy*$'$) blocks at most D other requests that could have been accepted instead. All requests in the optimal solution that are routed along a path of length at most D in the optimal solution must either be accepted by the algorithm or intersect a path accepted by the algorithm. Thus, we can apply the proof technique of the previous section to show that the optimal solution contains at most $(D+1)|\mathcal{A}|$ requests with paths of length at most D, where $\mathcal{A}$ is the set of

requests accepted by *BoundedGreedy′*. On the other hand, by the same argument as above, the number of paths in the optimal solution that are routed along paths of length at least $D+1$ is at most $m/(D+1)$. Thus, the optimal solution contains at most $m/(D+1) + (D+1)|\mathcal{A}| \leq (m/(D+1) + D + 1)|\mathcal{A}|$ paths. This proves that *BoundedGreedy′* achieves approximation ratio at most $D + 1 + m/(D+1)$. We can choose $D = \lceil\sqrt{m}\,\rceil - 1$ to obtain a $2\lceil\sqrt{m}\,\rceil$-approximation algorithm for MEDP.

To construct a bad instance for *BoundedGreedy′*, take a ring with m edges and vertices, the vertices being numbered clockwise from 1 to m. Choose m such that $\sqrt{m}$ is integral. Consider the set of requests containing the *first* request $(1, \sqrt{m})$, the *short* requests $(j, j+1)$ for $j = 1, \ldots, \sqrt{m}-1$, and the *long* requests $(i\sqrt{m}, (i+1)\sqrt{m})$ for $i = 1, \ldots, \sqrt{m}-1$. The algorithm accepts the first request $(1, \sqrt{m})$ and routes it along the clockwise path. All other requests cannot be routed along a path of length at most $D = \sqrt{m}-1$ now, so the algorithm rejects all other requests. On the other hand, the optimal solution accepts all short requests and all long requests, thus giving a solution with $2\sqrt{m} - 2$ requests.

Theorem 2 (Kleinberg, 1996). *Algorithm BoundedGreedy′ with parameter* $D = \lceil\sqrt{m}\,\rceil - 1$ *has approximation ratio* $O(\sqrt{m})$ *for* MEDP *in directed or undirected graphs with* m *edges.*

We will see later that *BoundedGreedy* achieves a much better approximation ratio in expanders and other specific classes of graphs. Furthermore, it is interesting to note that *BoundedGreedy′* can be adapted to the on-line setting by flipping a coin in the beginning and running *BoundedGreedy* with probability $\frac{1}{2}$ and the standard greedy algorithm otherwise.

Routing Number and Flow Number Kolman and Scheideler [40] analyze the approximation ratio achievable by the bounded-length greedy algorithm for undirected graphs in terms of the *routing number* of the given graph. Consider an undirected graph $G = (V, E)$ with n nodes. For a set P of paths in G, the congestion of an edge $e \in E$ is the number of paths containing e. The congestion of P is the maximum congestion among all edges. Denote by S_n the set of all permutations from V to V. For any permutation $\pi \in S_n$ and any L that is at least the diameter of G, let $C(G, L, \pi)$ be the minimum congestion among all path sets that realize π (i.e., contain a path from v to $\pi(v)$ for all $v \in V$) and that consist of paths with length at most L. Then the *L-bounded routing number* $R(G, L)$ of G is defined by

$$R(G, L) = \max_{\pi \in S_n} \max\{C(G, L, \pi), L\}\,.$$

Intuitively, if a graph has L-bounded routing number R, then for any permutation of the vertices, there exists a set of paths with length at most L and congestion at most R that realizes the permutation.

The *(unbounded) routing number* $R(G)$ is defined as $R(G) = \min_L R(G, L)$. For any graph G, $R(G)$ lies between $\Theta(\beta(G)^{-1})$ and $O(\Delta\beta(G)^{-1} \log n)$, where

Δ is the maximum degree of G and $\beta(G)$ is the expansion as defined in Section 1.1. There is a constant-factor approximation algorithm for computing the routing-number of a given graph [52]. The routing number is $\Theta(n)$ for the chain, $\Theta(\sqrt{n})$ for the $\sqrt{n} \times \sqrt{n}$-mesh, and $\Theta(\log n)$ for the butterfly, the hypercube, and constant-degree expanders.

For undirected graphs with maximum degree Δ and routing number R, Kolman and Scheideler show that the bounded-length greedy algorithm with length parameter $D = 2R$ achieves approximation ratio at most $(\Delta+4)R+1 = O(\Delta R) = O(\Delta^2\beta(G)^{-1}\log n)$. For graphs with bounded degree, this gives approximation ratio $O(R) = O(\beta(G)^{-1}\log n)$. If the exact value of R is not known, the bounded-length greedy algorithm can be run for all values $D = 2^p$ for $p = 0, 1, \ldots, \log n$, and the best solution can be taken as the output of the algorithm. Thus, approximation ratio $O(\Delta R)$ is achieved also in this case.

In addition, Kolman and Scheideler present a randomized approximation algorithm with ratio $O(\Delta\sqrt{LR})$ for MEDP in undirected graphs with maximum degree Δ and L-bounded routing number R [40]. The algorithm works in the online setting. Note that the bound $O(\Delta\sqrt{LR})$ can be substantially better than the bound $O(\Delta R(G))$. The idea behind the algorithm is that in many graphs, some edges are "bottleneck" edges and some are not. A single path passing through many bottleneck edges could cause the rejection of many subsequent requests. The algorithm thus puts a stricter bound on the number of bottleneck edges of a path than on the number of non-bottleneck edges. The tricky part of the algorithm is the initial phase in which the bottleneck edges are determined.

In subsequent work, Kolman and Scheideler [41] consider the *flow number* $F(G)$ of the given graph G instead of the routing number. They show that the bounded-length greedy algorithm with length parameter $D = 4F(G)$ achieves approximation ratio $O(F(G)) = O(\Delta\beta(G)^{-1}\log n)$ for MEDP in undirected graphs with maximum degree Δ and expansion $\beta(G)$.

It is interesting to note that $F(G)$ is 1 for complete graphs, $\Theta(\log n)$ for hypercubes and expanders, $\Theta(\sqrt{n})$ for the $\sqrt{n} \times \sqrt{n}$-mesh, and $\Theta(n)$ for the chain. As the flow number is defined in the more general context of graphs with edge capacities, we give its definition only in Section 5.1 where we use it in the context of the unsplittable flow problem.

The main idea underlying the use of the flow number is that the bounded-length greedy algorithm has a good approximation ratio if there exists an optimal solution that consists of short paths only. The key ingredient of the analysis is then a *shortening lemma*, which implies that any set of disjoint paths in a graph with flow number F can be converted into a set of fractional flows such that the load on every edge is at most 2 and all flow paths have length at most $4F$. The shortening lemma is applied to the paths in the optimal solution of the given instance of MEDP, and then the size of the solution produced by the bounded-length greedy algorithm is related to the fractional flows with short flow paths.

Kolman and Scheideler [41] use this analysis to obtain approximation ratio $16F + 1$ for the unsplittable flow problem (that includes MEDP as a special

```
algorithm ShortestFirstGreedy(G = (V, E), R = {(s_i, t_i) | i = 1, ..., k}):
  A ← ∅;
  while R contains a request that can be routed in G do
      (s_i, t_i) ← a request in R such that the shortest path from s_i to t_i in G
                   has minimum length among all requests in R;
      A ← A ∪ {(s_i, t_i)};
      R ← R \ {(s_i, t_i)};
      π_i ← a shortest path from s_i to t_i in G;
      remove all edges of π_i from G;
  od;
  return A and {π_i | (s_i, t_i) ∈ A};
```

Fig. 3. The shortest-path-first greedy algorithm.

case), and it is not difficult to verify that the ratio for MEDP can even be bounded by $8F + 1$ [13].

2.3 The Shortest-Path-First Greedy Algorithm

Another modification of the greedy algorithm is the shortest-path-first greedy algorithm, shown in Fig. 3. It was suggested by Kolliopoulos and Stein [37]. Instead of processing the given requests sequentially in an arbitrary order, the algorithm *ShortestFirstGreedy* considers all remaining requests and accepts a request whose shortest path has length ℓ only if no other remaining request can be routed along a path of length smaller than ℓ. As a consequence, the algorithm accepts requests in order of non-decreasing lengths of their shortest paths. It is easy to see that the worst-case approximation ratio of *ShortestFirstGreedy* is at least as good as that of *BoundedGreedy*.

Algorithm *ShortestFirstGreedy* achieves approximation ratio $\lceil\sqrt{m}\,\rceil$. To prove this, consider an arbitrary optimal solution $\mathcal{A}^*$ with paths π_i^*. Again, consider an execution of the algorithm and when the algorithm accepts a request i and routes it along a path π_i, remove from $\mathcal{A}^*$ the request i (if $i \in \mathcal{A}^*$) as well as all requests j whose path π_j^* intersects π_i. By the order in which the algorithm accepts the requests, all requests $j \in \mathcal{A}^*$ have length at least $|\pi_i|$ when the algorithm accepts request i. If $|\pi_i| \leq \lceil\sqrt{m}\,\rceil - 1$, we have to remove at most $\lceil\sqrt{m}\,\rceil$ paths from $\mathcal{A}^*$, as before. If $|\pi_i| \geq \lceil\sqrt{m}\,\rceil$, there can be at most $\sqrt{m}$ paths left in $\mathcal{A}^*$, because they all have length at least $\lceil\sqrt{m}\,\rceil$. So we have to remove at most $\sqrt{m}$ paths in this case as well. This shows that the approximation ratio of *ShortestFirstGreedy* is at most $\lceil\sqrt{m}\,\rceil$.

Theorem 3 (Kolliopoulos and Stein, 1998). *The shortest-path-first greedy algorithm has approximation ratio at most* $\lceil\sqrt{m}\,\rceil$ *for* MEDP *in directed or undirected graphs with* m *edges.*

We remark that m can be replaced by m^* in the analysis above, where m^* is the number of edges that are actually used by paths in the optimal solution.

By refining the analysis above, we can bound the approximation ratio of *ShortestFirstGreedy* in terms of the average path length $d^* = m^*/|\mathcal{A}^*|$ of an optimal solution $\mathcal{A}^*$ whose paths use m^* edges in total. Let $\mathcal{A}' = \mathcal{A}^* \setminus \mathcal{A}$, i.e., $\mathcal{A}'$ denotes the requests that are in the optimal solution, but not in the solution computed by the algorithm. When the algorithm accepts request i and routes it along path π_i, we remove from $\mathcal{A}'$ the requests whose paths π_j^* intersect π_i. Let k_i denote the number of requests that are removed from $\mathcal{A}'$ because of path π_i. Each of these k_i requests has a path of length at least $|\pi_i|$, and $|\pi_i|$ must be at least k_i. Therefore, these k_i requests use at least k_i^2 of the m^* edges used in the optimal solution, and we get

$$\sum_{i \in \mathcal{A}} k_i^2 \leq m^*.$$

Since $\sum_{i \in \mathcal{A}} k_i^2 \geq \left(\sum_{i \in \mathcal{A}} k_i \right)^2 / |\mathcal{A}|$ by the Cauchy-Schwarz inequality, we get $\left(\sum_{i \in \mathcal{A}} k_i \right)^2 / |\mathcal{A}| \leq m^*$. Using $\sum_{i \in \mathcal{A}} k_i = |\mathcal{A}'|$, we obtain $|\mathcal{A}| \geq |\mathcal{A}'|^2/m^*$. If $|\mathcal{A}| \geq |\mathcal{A}^*|/2$, the algorithm has ratio at most 2. If $|\mathcal{A}| < |\mathcal{A}^*|/2$, we have $|\mathcal{A}'| = |\mathcal{A}^* \setminus \mathcal{A}| > |\mathcal{A}^*|/2$ and obtain

$$|\mathcal{A}| \geq \frac{|\mathcal{A}'|^2}{m^*} > \frac{|\mathcal{A}^*|^2}{4m^*} = \frac{|\mathcal{A}|^*}{4\frac{m^*}{|\mathcal{A}^*|}} = \frac{|\mathcal{A}|^*}{4d^*}.$$

Thus, the approximation ratio is always bounded by $4d^*$.

Theorem 4 (Kolliopoulos and Stein, 1998). *The shortest-path-first greedy algorithm achieves approximation ratio $O(d^*)$ for* MEDP *on instances for which some optimal solution has average path length d^*.*

We remark that approximation ratio $O(d^*)$ can also be achieved by running *BoundedGreedy* with parameter $D = 1, 2, 4, 8, \ldots, n$ and outputting the best of the solutions obtained in this way. This follows because at least half the paths of the optimal solution have length at most $2d^*$, and *BoundedGreedy* will produce a solution with the required quality for the value of D that lies in the interval $[2d^*, 4d^*)$ (see the full version of [40]).

Chekuri and Khanna [15] showed that the bound of $O(\sqrt{m})$ on the approximation ratio of the shortest-path-first greedy algorithm is not tight for dense graphs. They proved upper bounds of $O(n^{2/3})$ and $O(n^{4/5})$ for undirected and directed graphs, respectively. Their improved result is obtained by tightening the analysis concerning long paths in the optimal solution. In the arguments above, we used the fact that the optimal solution can contain at most m/ℓ paths of length at least ℓ. Roughly speaking, Chekuri and Khanna show that in a graph where the distance between the end vertices of each request is at least ℓ, at most $O(n^2/\ell^2)$ (in the undirected case) and $O(n^4/\ell^4)$ (in the directed case) requests can be connected along edge-disjoint paths. This leads to the better bound in dense graphs. They also show that the approximation ratio of the shortest-path-first greedy algorithm is $\Omega(n^{2/3})$ in directed acyclic graphs and in undirected graphs. The upper bound on the approximation ratio of the shortest-path-first greedy algorithm for directed graphs was subsequently improved to

$O((n \log n)^{2/3})$ by Varadarajan and Venkataraman [56]. They obtain this result by showing that in a directed graph where the distance between the end vertices of each request is at least ℓ, at most $O((n^2/\ell^2) \cdot (\log n/\ell)^2)$ of the requests can be connected along edge-disjoint paths.

Theorem 5 (Chekuri and Khanna, 2003; Varadarajan and Venkataraman, 2004). *The approximation ratio of the shortest-path-first greedy algorithm for* MEDP *is* $O(\min\{\sqrt{m}, n^{2/3}\})$ *in undirected graphs and* $O(\min\{\sqrt{m}, (n \log n)^{2/3}\})$ *in directed graphs.*

For directed acyclic graphs, Chekuri and Khanna [15] present a combinatorial algorithm that achieves approximation ratio $O(\sqrt{n} \cdot \log n)$. The algorithm is based on the observation that, if the optimal solution contains many long paths, there must be a vertex that is contained in a significant fraction of these long paths.

2.4 Inapproximability for Directed Graphs

The bounded greedy algorithm and the shortest-path-first greedy algorithm achieve approximation ratio $O(\sqrt{m})$ for directed or undirected graphs with m edges. For directed graphs, an essentially matching inapproximability result has been obtained by Guruswami et al. [31].

Theorem 6 (Guruswami et al., 1999). *For* MEDP *in directed graphs with* m *edges, there cannot be an* $m^{\frac{1}{2}-\varepsilon}$*-approximation algorithm for any* $\varepsilon > 0$ *unless* $\mathcal{P} = \mathcal{NP}$*.*

Proof. The proof is based on the hardness of the problem 2DirPath, i.e., deciding for a directed graph $H = (V, E)$ and four distinct vertices u_1, w_1, u_2, w_2 whether H contains paths from u_1 to w_1 and from u_2 to w_2 that are edge-disjoint. This problem has been proved $\mathcal{NP}$-complete by Fortune, Hopcroft and Wyllie in [23]. Consider some fixed $\varepsilon > 0$. Given an instance I of the problem 2DirPath, one can efficiently construct an instance of MEDP with k requests in a directed graph G with m edges such that the optimal solution contains all k requests if I is a yes-instance and only one request if I is a no-instance. Therefore, an approximation algorithm for MEDP with ratio smaller than k would allow to decide I in polynomial time, implying that $\mathcal{P} = \mathcal{NP}$.

The construction of G is illustrated in Fig. 4. G is the lower right triangle of a $k \times k$ mesh, with all edges pointing upward or to the right. Every internal vertex v of the mesh is replaced by a copy of H. The head of the incoming horizontal edge of v is attached to u_1, the tail of the outgoing horizontal edge to w_1. Similarly, the vertical edges incident to v are attached to u_2 and w_2. Intuitively, this copy of H allows two paths arriving at v from below and from the left to *cross* (i.e., to leave this copy of H on the top and on the right side, respectively) if and only if I is a yes-instance.

The k requests in G are defined as follows: The vertices at the bottom of the mesh (from left to right) are the sources s_1, s_2, ..., s_k, and the vertices on the right side of the mesh (from bottom to top) are the destinations t_1, t_2,

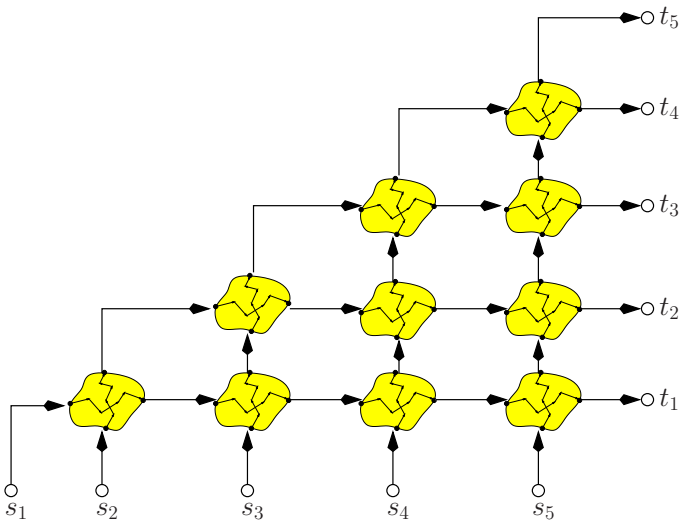

Fig. 4. Construction used in the inapproximability proof.

$\ldots, t_k$. If I is a yes-instance, all k requests can be accepted and routed along the canonical path that goes upward until the destination row is reached and then to the right until the destination is hit. If I is a no-instance, there cannot be edge-disjoint paths for any two requests i and j, $i \neq j$, because such paths would have to cross at some copy of H. Therefore, the optimal solution to the constructed instance of MEDP contains either k requests or only one request, depending on whether I is a yes-instance. If we choose $k = m_H^{1/\varepsilon}$, where m_H is the number of edges of H, the graph G has $m = \Theta(k^2 m_H) = \Theta(k^{2+\varepsilon})$ edges, hence $k = \Theta(m^{1/(2+\varepsilon)})$. Since the construction can be applied for every fixed $\varepsilon > 0$, the theorem follows. □

Note, however, that for the graphs constructed in the proof of Theorem 6, we have $m = O(n)$. Therefore, the lower bound does not necessarily apply to dense graphs. In fact, we have seen in Theorem 5 that the shortest-path-first greedy algorithm indeed achieves approximation ratio strictly better than $O(\sqrt{m})$ in dense graphs. In terms of n, the proof of Theorem 6 gives only $n^{\frac{1}{2}-\varepsilon}$-inapproximability of MEDP in directed graphs.

Another inapproximability result for MEDP in directed graphs has been obtained by Ma and Wang [45]. They prove that MEDP cannot be approximated within ratio $2^{\log^{1-\varepsilon} n}$ in directed graphs with n vertices unless $\mathcal{NP} \subseteq \mathrm{DTIME}(2^{\mathrm{polylog} n})$.

It remains an interesting open problem to determine whether the approximation ratio for MEDP in undirected graphs can be improved substantially. The proof of Theorem 6 cannot be adapted to the undirected case because the variant of 2DirPath for undirected graphs can be solved in polynomial time. In fact, for every constant k there is a polynomial-time algorithm that decides for an

undirected graph and k requests whether all k requests can be connected along edge-disjoint paths by the results of Robertson and Seymour [51]. At present, we cannot even exclude the existence of an $O(1)$-approximation algorithm for MEDP in undirected graphs.

2.5 Linear Programming and Randomized Rounding

Linear programming [53] is a very powerful tool in combinatorial optimization. In this section we discuss the natural LP relaxation of MEDP. Intuitively, solutions to the LP relaxation correspond to *fractional* solutions, i.e., solutions that may accept only fractions of certain requests and that may split the accepted fraction of a request among several paths arbitrarily.

In the following, we allow a more general edge capacity constraint than the strict requirement of having edge-disjoint paths: we assume that each edge e has a certain integral capacity $u(e)$ and that a routing is feasible if at most $u(e)$ paths go through every edge e. We call this problem *generalized MEDP*.

An LP Relaxation of MEDP Let a graph $G = (V, E)$ with edge capacities $u : E \to \mathbb{N}$ and a set of requests $\mathcal{R} = \{(s_i, t_i) \mid i = 1, \ldots, k\}$ be given. Denote by $\mathcal{P}_i$ the set of all paths from s_i to t_i in G. Note that there might be exponentially many paths in $\mathcal{P}_i$; we will discuss later how to deal with this.

It is natural to introduce a variable x_i for each request i, $1 \leq i \leq k$, that is constrained by $0 \leq x_i \leq 1$ and represents the fraction of request i that is accepted. Furthermore, a variable $y_{i,\pi}$ is introduced for each path $\pi \in \mathcal{P}_i$; this variable corresponds to the fraction of request i that is routed along path π.

Thus, maximizing the sum of the accepted fractions of the given requests can be formulated as the following linear program:

$$f^* = \max \sum_{i=1}^{k} x_i \tag{1}$$

$$\text{s.t.} \sum_{i,\pi : e \in \pi \in \mathcal{P}_i} y_{i,\pi} \leq u(e), \qquad \text{for all } e \in E \tag{2}$$

$$\sum_{\pi \in \mathcal{P}_i} y_{i,\pi} = x_i, \qquad \text{for } 1 \leq i \leq k \tag{3}$$

$$0 \leq x_i \leq 1, \qquad \text{for } 1 \leq i \leq k \tag{4}$$

$$0 \leq y_{i,\pi} \leq 1, \qquad \text{for } 1 \leq i \leq k \text{ and } \pi \in \mathcal{P}_i \tag{5}$$

The objective function (1) expresses that the goal is to maximize the sum of the accepted fractions of the requests. Constraint (2) ensures that the edge capacities are not exceeded. Constraint (3) requires that the fractions of request i that are routed along different paths from s_i to t_i add up to x_i, the total accepted fraction of request i. Constraints (4) and (5) ensure that all variables are between zero and one.

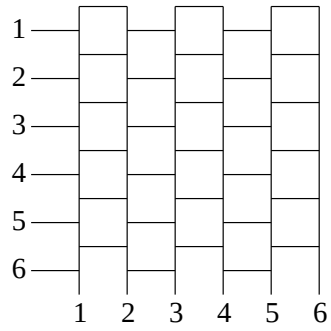

Fig. 5. The brick-wall graph.

The above LP may be of exponential size, but is easy to understand. In order to obtain an LP of polynomial size, one uses variables $f_{i,e}$ for $1 \leq i \leq k$ and $e \in E$ that represent the fraction of request i that is routed through edge e. The number of such variables is bounded by $k|E|$. Furthermore, the edge capacity constraints (2) are replaced by $\sum_{i=1}^{k} f_{i,e} \leq u(e)$ for all $e \in E$, and flow conservation constraints are added for each request i and each vertex $v \in V \setminus \{s_i, t_i\}$ (expressing that the amount of request i reaching v through incoming edges is equal to the amount leaving v through outgoing edges). Constraint (3) is replaced by a constraint ensuring that the amount of request i leaving s_i is equal to x_i. The variables $y_{i,\pi}$ are not needed in this modified LP. The modified LP has polynomial size and can be solved optimally in polynomial time. We refer to the resulting values of the variables $f_{i,e}$ and x_i as $f^*_{i,e}$ and x^*_i, respectively. These quantities can be viewed as representing a *flow* of value x^*_i from s_i to t_i, for $1 \leq i \leq k$. Using standard flow decomposition methods (also called path stripping) [50], the flow of request i can be efficiently transformed into $O(|E|)$ separate flows along paths from s_i to t_i, and these flows represent a fractional solution to the original (exponential-size) LP with the same objective value. Thus, we can assume from now on that we can compute an optimal fractional solution to the original LP in polynomial time and that this solution has only a polynomial number of variables $y^*_{i,\pi}$ that are non-zero.

A solution to the LP in which the variables x_i and $y_{i,\pi}$ are all integral, i.e., either equal to 0 or equal to 1, corresponds to a feasible solution of the original instance of generalized MEDP. However, adding such integrality constraints makes the linear programming problem $\mathcal{NP}$-hard. Therefore, a meaningful approach is to solve the LP to obtain an optimal fractional solution and then try to convert the fractional solution into an integral solution without losing too much in the objective function. This conversion is usually called *rounding*.

The Integrality Gap Unfortunately, the gap between the fractional optimum and the integral optimum can be arbitrarily large for the LP formulation of MEDP [28]. A simple example to demonstrate this is shown in Fig. 5. This type

of mesh-like graph with internal vertices of degree 3 is often called a *brick-wall graph*. Assume that the graph is undirected, all edges have capacity 1, and there are 6 requests, where both endpoints of request i are labeled by i in the figure. Since any two paths for different requests must cross somewhere and thus share an edge, the integral optimum accepts only one of the six paths. The fractional solution, however, can route 1/2 of each request without violating any capacity constraint, thus giving an objective value of 3. This example can be generalized to give instances where the integral optimum is 1, but the fractional optimum is $\Theta(\sqrt{m})$ for brick-wall graphs with m edges, where m is arbitrarily large. The construction can also be adapted to directed graphs in a straightforward way. Note, however, that the brick-wall graph is sparse and thus the lower bound on the integrality gap in terms of n is only $\Omega(\sqrt{n})$. Chekuri and Khanna [15] prove that an upper bound of $O(\sqrt{n} \cdot \log n)$ on the integrality gap holds for directed, acyclic graphs.

In any case, the discussion above shows that we cannot hope to find a rounding approach that always gives an integral solution with objective value close to the fractional optimum.

Randomized Rounding in the High-Capacity Case Fortunately, the situation is much better if the edge capacities are required to be large. In this case, the *randomized rounding* approach due to Raghavan and Thompson [50] gives a constant-factor approximation ratio. Our presentation follows the work by Kleinberg [33, pp. 39–41]. The basic idea is to interpret the values of the variables in the fractional solution as probabilities. For the variables x_i, this means that x_i is set to 1 with probability x_i^*. More precisely, we will have to scale down the values x_i^* by some factor μ before the rounding can be done.

Assume that all edge capacities are at least $\varepsilon \log m$ for some constant ε. Let $\mu < 1$ be a positive constant whose choice will be explained later. For each request i, $1 \leq i \leq k$, make an independent random choice and route request i along path $\pi \in \mathcal{P}_i$ with probability $\mu y_{i,\pi}^*$, for all non-zero $y_{i,\pi}^*$, and reject request i with probability $1 - \sum_{\pi \in \mathcal{P}_i} \mu y_{i,\pi}^* = 1 - \mu x_i^*$.

We need to show that with constant probability, the resulting integral solution does not violate any edge capacity and that its objective value is at least a constant fraction of the fractional optimum. The number of paths going through an edge in the integral solution is the number of fractional paths through that edge that were rounded up. Thus, the random variable representing the number of paths going through the edge in the integral solution can be viewed as the sum of independent Bernoulli trials; since the number of trials is large in the case of large edge capacities, Chernoff-Hoeffding bounds [47] can be used to show that the probability that some edge capacity is violated is at most $1/m$ if μ is chosen as $\mu = e^{-1}4^{-1/\varepsilon}$. Furthermore, by the Markov inequality, the probability that the objective value resulting from the rounding is at least $\mu/2$ times the fractional optimum is at least $\mu/(2-\mu)$. Therefore, the probability that no edge capacity is violated and the objective value is at least $\mu/2$ times the fractional optimum is a constant. Thus, repeating the randomized rounding $\Theta(p)$ times

provides a constant-factor approximation algorithm for the high-capacity case of MEDP with probability $1 - 2^{-p}$.

Theorem 7 (Raghavan and Thompson, 1987; Kleinberg, 1996). *For all instances of generalized* MEDP *where the edge capacities are at least* $\varepsilon \log m$ *for a suitable constant* ε*, there is a randomized polynomial-time algorithm that achieves a constant-factor approximation with high probability.*

Note that the constant-factor approximation ratio of this theorem holds also if the solution of the algorithm is compared with the *fractional* optimum of the LP relaxation of the problem.

Kolman and Scheideler [41] prove that a constant-factor approximation algorithm for generalized MEDP (in fact, even for the unsplittable flow problem) in graphs with flow number F exists if the edge capacities are at least $\varepsilon \log F$ for a suitable constant ε. Their result is obtained by a rather sophisticated use of randomized rounding. Note that for graphs with polylogarithmic flow number (hypercubes, expanders, etc.), this shows that constant approximation ratio can already be obtained if the edge capacities are $\Omega(\log \log n)$. This latter result had previously been obtained by Srinivasan [55] (see also Baveja and Srinivasan [5]).

Further results about the approximation ratio achieved by LP-based algorithms will be discussed in Section 5.5 in the context of the unsplittable flow problem.

3 Edge-Disjoint Paths in Specific Graph Classes

Since the approximation ratios that have been achieved for arbitrary graphs are quite large, it is interesting to investigate whether MEDP admits better approximation ratios for restricted classes of graphs. There are two basic techniques that sometimes allow to obtain a better approximation ratio on specific graph classes:

- For some graph classes, the approximation ratio of the standard greedy algorithm of Section 2.1 can be improved by sorting the given requests in some order depending on the structure of the given graph.
- For some graph classes, one can prove that there exists a solution using paths of length at most D that is at least as large as $1/\alpha$ times the optimum solution. The bounded-length greedy algorithm of Section 2.2 with parameter D and the shortest-path-first greedy algorithm of Section 2.3 both give approximation ratio at most αD in this case (cf. Kolman [38]).

We will see examples of both techniques in the following.

3.1 Trees and Trees of Rings

Recall that MEDP can be solved optimally in polynomial time for undirected trees, but is $\mathcal{APX}$-hard for bidirected trees and undirected or bidirected trees

of rings. A simple approximation algorithm for MEDP in bidirected trees is the following: Root the given tree at an arbitrary node. For each node v of the tree, let $d(v)$ denote its distance from the root. For each request (s_i, t_i), let ℓ_i be the node on the path from s_i to t_i that is closest to the root. The node ℓ_i is also called the *lowest common ancestor* of s_i and t_i. Now process the given requests in order of non-increasing values $d(\ell_i)$ and apply the greedy algorithm (i.e., accept each path if it does not intersect any previously accepted path).

In order to analyze the algorithm, we fix an arbitrary optimal solution $\mathcal{A}^*$. When the algorithm accepts a request (s_i, t_i), we remove from $\mathcal{A}^*$ all paths that intersect (s_i, t_i). By definition of the algorithm, all paths $j \in \mathcal{A}^*$ have $d(\ell_j) \leq d(\ell_i)$. Therefore, if such a path $j \in \mathcal{A}^*$ intersects the path π_i from s_i to t_i, it must contain one of the (at most) two edges on π_i that are incident to ℓ_i. This means that there can be at most two paths in $\mathcal{A}^*$ that intersect π_i. Thus, the algorithm achieves approximation ratio 2.

An improved approximation algorithm with ratio $\frac{5}{3} + \varepsilon$, where $\varepsilon > 0$ is an arbitrary fixed constant, was presented by Erlebach and Jansen [21]. This algorithm is based on the simple 2-approximation just described, but it considers all paths with the same lowest common ancestor simultaneously and computes a maximum number s of such paths that can be added to the current solution. This can be done using a maximum bipartite matching algorithm. If $s \geq 3$, all these s paths are accepted. Since it suffices to remove from $\mathcal{A}^*$ at most s paths with the same lowest common ancestor and at most two other paths whose lowest common ancestor is closer to the root, at most $\frac{5}{3}s$ paths have to be removed from $\mathcal{A}^*$. If $s = 1$ or $s = 2$, the algorithm cannot always make the decision regarding acceptance or rejection right away. In some cases, it creates a configuration of *unresolved paths*, i.e., paths which are neither accepted nor rejected and whose status will be decided at a later node. Such configurations of unresolved paths complicate the processing at later nodes, but a careful case analysis shows that approximation ratio $\frac{5}{3} + \varepsilon$ can be achieved for any $\varepsilon > 0$. The running-time is polynomial in the size of the input for fixed $\varepsilon > 0$.

Theorem 8 (Erlebach and Jansen, 2001). *For every fixed $\varepsilon > 0$, there is a $(\frac{5}{3} + \varepsilon)$-approximation algorithm for* MEDP *in bidirected trees.*

An example of a tree of rings is shown in Fig. 6. In undirected or bidirected trees of rings, it is possible to reduce the problem to that for trees while losing at most a factor of 3 in the approximation ratio. The idea is just to remove one link from every ring of the tree of rings arbitrarily. This turns the tree of rings into a tree, and at least one third of the requests in the optimal solution for the tree of rings can still be routed in the tree. Thus, in the undirected case one can apply the optimal algorithm for undirected trees, leading to approximation ratio 3 for MEDP in undirected trees of rings. For bidirected trees of rings, approximation ratio $5 + \varepsilon$ is obtained by using the $(\frac{5}{3} + \varepsilon)$-approximation algorithm for MEDP in bidirected trees of Theorem 8.

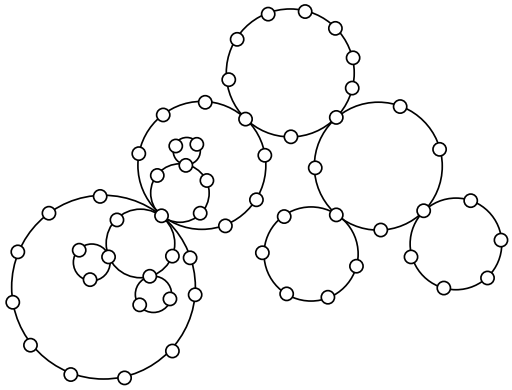

Fig. 6. A tree of rings.

Theorem 9 (Erlebach, 2001). *There exists a* 3*-approximation algorithm for* MEDP *in undirected trees of rings and a* $(5+\varepsilon)$*-approximation algorithm for* MEDP *in bidirected trees of rings, for every fixed* $\varepsilon > 0$*.*

3.2 Meshes and Densely Embedded Graphs

Kleinberg and Tardos [36] found a randomized $O(1)$-approximation algorithm for MEDP in two-dimensional meshes. Furthermore, they generalized the algorithm to obtain approximation ratio $O(1)$ also for the class of *densely embedded, nearly-Eulerian graphs*; we refer to [36] for details. In the following, we give an outline of their algorithm for meshes.

Fix some constant γ. The given requests are partitioned into *long requests* and *short requests*. A request (s_i, t_i) is *short* if the distance between its endpoints is at most $16\gamma \log n$, and *long* otherwise. By considering short requests and long requests separately, computing an approximate solution for each of the two sets, and outputting the better of the two, an algorithm loses at most a factor of 2 in the approximation ratio. This is because at least half the requests in an optimal solution are either all short or all long.

The basic idea for dealing with the long requests is to embed a simulated network with high capacity into the mesh and to use an $O(1)$-approximation algorithm for high-capacity networks based on linear programming and randomized rounding (cf. Section 2.5) in this simulated network. One difficulty is translating the accepted paths in the simulated network into edge-disjoint paths in the original mesh.

The short requests are again partitioned into *medium requests* and *small requests*. The medium requests will be handled by applying the algorithm for long requests in separate submeshes of the original mesh. The small requests are short enough to be handled by brute-force enumeration.

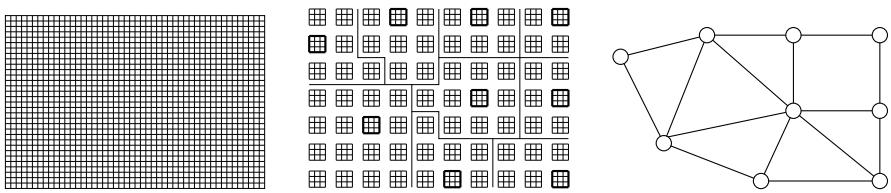

Fig. 7. The given mesh (left); its partitioning into $\gamma \log n \times \gamma \log n$ subsquares, with chosen subsquares drawn in bold and additional lines showing the border of the enclosures (middle); and the core of the resulting simulated network (right).

Long Requests and the Simulated Network For dealing with long requests, the mesh is partitioned into subsquares of size $\gamma \log n \times \gamma \log n$. The subsquare whose middle vertex is v is denoted by C_v. For a subsquare C_v, the up to eight other subsquares that are adjacent to it are called its *neighbors*. A maximal subset of the subsquares is chosen randomly such that no two chosen subsquares have a common neighbor. The random choice ensures that every subsquare becomes a chosen subsquare with constant probability and for any pair of subsquares whose middle vertices are at distance at least $11\gamma \log n$, there is a constant probability that both subsquares are chosen. The algorithm discards all requests that do not have both endpoints in chosen subsquares. This costs only a constant factor in the approximation ratio.

A chosen subsquare C_v together with all its neighbors becomes an *enclosure* D_v. Subsquares that do not belong to an enclosure at this point are included in any of their adjacent enclosures arbitrarily. Thus, the vertices of the mesh are partitioned into enclosures. Two enclosures D_v and D_w are adjacent if there is an edge with one endpoint in D_v and the other endpoint in D_w. In this way, an enclosure can be adjacent to no more than 20 other enclosures. See Fig. 7 for an example.

Now the simulated network is built. First, take a vertex z_v for each chosen subsquare C_v, and connect two vertices z_v and z_w if their enclosures are adjacent. This defines the core $\mathcal{N}$ of the simulated network, and all edges in this core will be assigned capacity $\rho \log n$ for a constant $\rho < \gamma$. The right-hand side of Fig. 7 depicts the core of the simulated network in the example. To obtain the final simulated network $\mathcal{N}'$, add a copy of each chosen subsquare C_v and make all vertices at the boundary of C_v adjacent to z_v. The edges inside C_v and the edges from the boundary of C_v to z_v are assigned capacity 1.

The long requests with endpoints in chosen subsquares can be viewed as requests in $\mathcal{N}'$. Formulating the problem in $\mathcal{N}'$ as a linear program and solving this linear program, we obtain an optimal fractional solution F^* for the problem. The objective value of this fractional solution is within a constant factor of the optimal integral solution for the long requests in the mesh. By applying the randomized rounding approach of Raghavan and Thompson, we obtain an integral solution F (a set of accepted requests and one path for each accepted request) that, with high probability, does not violate any edge capacity in $\mathcal{N}$

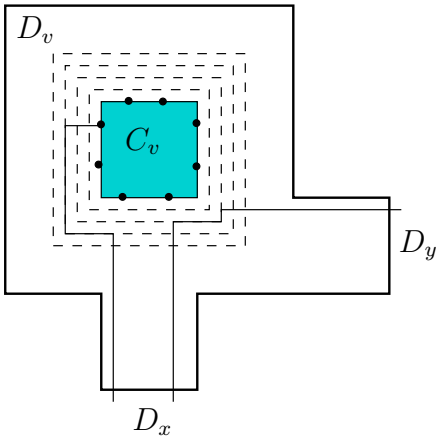

Fig. 8. The $\frac{1}{2}\gamma \log n$ rings outside C_v realize a crossbar structure in D_v. Two paths using the crossbar structure are shown: one path arrives from D_x and is routed to a vertex on the boundary of C_v, the other arrives from D_x and is routed to D_y.

and whose cardinality is a constant fraction of the fractional optimum. However, the paths in F may violate edge capacities in C_v.

The paths of the integral solution F in $\mathcal{N}'$ must be translated back into paths in the mesh. For each pair of adjacent enclosures D_v and D_w, choose a set $\tau_{v,w}$ of $\rho \log n$ edges connecting D_v and D_w arbitrarily. Furthermore, choose a set σ_v of $\rho \log n$ vertices equally spaced at the boundary of each C_v. The part of the mesh around a chosen subsquare C_v will be used to establish a crossbar structure allowing arbitrary interconnection of vertices in σ_v and endpoints of edges in all sets $\tau_{v,w}$ along edge-disjoint paths. The up to $\rho \log n$ paths going from z_v to z_w in $\mathcal{N}'$ will be routed through edges of $\tau_{v,w}$, and up to $\rho \log n$ paths with an endpoint s in C_v are to be routed from s to some vertex in σ_v.

To establish the crossbar structure, the $\frac{1}{2}\gamma \log n$ rings induced by vertices at distance $\gamma \log n$ to $\frac{3}{2}\gamma \log n$ from the middle vertex v of a chosen subsquare C_v are used. This is illustrated in Fig. 8. Each vertex in σ_v and each endpoint in D_v of an edge in some $\tau_{v,w}$ is assigned a different such ring; since D_v is adjacent to at most 20 other enclosures, this is possible if ρ is chosen smaller than $\gamma/42$. The outermost $\rho \log n$ rings are assigned to vertices in σ_v. It is easy to see that we can assign each vertex in σ_v and each endpoint in D_v of an edge in some $\tau_{v,w}$ a path from that vertex to its ring, such that all these paths are edge-disjoint and do not use an edge of any of the rings.

Consider a path π_F assigned to request (s_i, t_i) in F. Let s_i and t_i be contained in C_v and C_w, respectively. To route (s_i, t_i) in the mesh, a path is obtained as follows:

– Find a path from s_i to some vertex $r_{i,v}$ in σ_v and a path from t_i to some vertex $r_{i,w}$ in σ_w. These paths form the *first segment* and *last segment* of the constructed path from s_i to t_i in the mesh.

– If z_u is the first vertex after z_v in π_F, choose a free edge e_i in $\tau_{v,u}$, and let its endpoint in D_v be u_i. Follow the path from $r_{i,v}$ to its ring in D_v until it intersects the ring of u_i, then follow that ring and the path from that ring to u_i. Then take the edge e_i.
– Consider an intermediate vertex z_u with successor z_x on π_F. If the constructed path enters D_u through an edge e_1, pick a free edge e_2 in $\tau_{u,x}$. Assume without loss of generality that the ring of e_1 in D_u is inside the ring for e_2. Follow the path from e_1 to its ring until the ring of e_2 is hit, then follow the ring of e_2 and the path to e_2.
– At the last $\mathcal{N}$-vertex z_w of π_F (i.e., the last vertex of π_F that is in the core $\mathcal{N}$ of the simulated network $\mathcal{N}'$), if the partial path enters D_w through edge e_1, follow the path from e_1 to its ring in D_w until it intersects the ring of $r_{i,w}$, then follow that ring and its path to $r_{i,w}$.

Except for the first and last segment of the paths, it is clear that we find edge-disjoint paths in the mesh for each path in F.

It remains to deal with the first and last segments. Consider some subsquare C_v. The solution F can have more paths with an endpoint in C_v than can be routed along edge-disjoint paths to distinct vertices in σ_v. Kleinberg and Tardos model the *escape problem* of routing path endpoints in C_v to vertices in σ_v as a flow problem and observe that an edge-disjoint routing exists provided the cut condition is satisfied for all rectangular cuts. For the case that the cut condition is violated by the requests accepted in the solution F, they show that one can discard requests from F so that the remaining requests are a constant fraction of the requests in F and all cut conditions are satisfied. For the remaining requests, a network flow algorithm can be used to solve the escape problem and to obtain the first and last segments of the paths as required. In total, this gives an $O(1)$-approximation algorithm for the long requests.

Short Requests Recall that the endpoints of short requests are at distance smaller than $16\gamma \log n$. Set $r = 32\gamma \log n$. First, the algorithm randomly chooses a maximal set M of vertices in the mesh with the property that the L_∞ distance between any two vertices in M is more than $4r$. For $u \in M$, let X_u be the set of all vertices with L_∞ distance at most r from u, and let Y_u be the set of all vertices with L_∞ distance at most $2r$ from u. Note that Y_u and $Y_{u'}$ are disjoint for $u, u' \in M$, $u \neq u'$. If M is chosen by an appropriate randomized algorithm, it can be shown that for any short request (s_i, t_i), there is a constant probability that both s_i and t_i are contained in X_u for some $u \in M$. Thus, the algorithm can restrict its attention to short requests with both endpoints in the same set X_u for some $u \in M$ and discard all other short requests. Furthermore, the optimal solution for the short requests with endpoints in X_u using paths in the submesh induced by Y_u can be shown to route at least one quarter of the requests in an optimal solution for these requests using paths in the whole mesh. Thus, the algorithm can deal with the requests in each X_u separately and use only the submesh induced by Y_u to route the requests with both endpoints in X_u.

Now consider the short requests with both endpoints in X_u. By considering these requests as requests in the submesh induced by Y_u, we can recursively partition them into long requests and short requests with respect to the submesh. Call the resulting long requests *medium requests* and the resulting short requests *small requests*. The medium requests are handled by the algorithm for long requests of the previous section, now applied to the submesh induced by Y_u. The small requests have endpoints whose distance is $O(\log \log n)$. By selecting subsets X'_v and Y'_v inside Y_u in the same way as X_u and Y_u were selected in the original mesh, the routing of small requests is reduced to solving MEDP in submeshes with side length $O(\log \log n)$. These submeshes are so small that an optimal solution can be computed by a brute-force enumeration in polynomial time.

Thus, we get approximation algorithms with constant ratio for the long requests, medium requests, and small requests. By running all three algorithms and outputting the largest of the three solutions, we get a constant-factor approximation algorithm for MEDP in meshes. Since some of the steps hold only with constant probability, parts of the algorithm must be repeated a certain number of times in order to guarantee that the constant-factor approximation is achieved with high probability.

Theorem 10 (Kleinberg and Tardos, 1995). *For* MEDP *in undirected meshes there is a randomized polynomial-time approximation algorithm that achieves a constant-factor approximation with high probability.*

Prior to this result, an $O(\log n \log \log n)$-approximation algorithm (that was an on-line algorithm) for meshes had been obtained by Awerbuch et al. [3] and $O(\log n)$-approximation algorithms by Aumann and Rabani [1] and Kleinberg and Tardos [35]. Kleinberg and Tardos [36] presented also an on-line algorithm with approximation ratio $O(\log n)$ for MEDP in meshes.

3.3 Hypercubes and de Bruijn Graphs

MEDP admits an $O(\log n)$-approximation algorithm in undirected or bidirected hypercubes. For bidirected hypercubes, Gu and Tamaki [29] have shown that any partial permutation can be realized with two sets of edge-disjoint paths. Taking the larger of the two sets gives a 2-approximation algorithm for MEDP on partial permutation instances. For general instances, we can first compute a largest subset of the given requests that is a partial permutation instance and then use the 2-approximation algorithm by Gu and Tamaki. By reducing a general instance to a partial permutation instance, we lose at most a factor of $O(\log n)$ in the approximation ratio, because the vertices of the hypercube have degree $d = \log n$. For undirected hypercubes, it was shown by Aumann and Rabani in [1] that any partial permutation can be realized by a constant number of sets of edge-disjoint paths. Hence, the same reasoning can be applied to undirected hypercubes as well. Since hypercubes have flow number $O(\log n)$, it follows from the work of Kolman and Scheideler [41] that *BoundedGreedy*

achieves ratio $O(\log n)$ for MEDP in hypercubes. Hence, this approximation ratio can even be achieved in the on-line setting.

An embedding of the butterfly into the de Bruijn graph and a decomposition of the de Bruijn graph were presented by Kolman [38]. From these he derived, for every constant $\varepsilon > 0$, an $O(\log^{2+\varepsilon} n)$-approximation algorithm for MEDP in undirected de Bruijn graphs. As de Bruijn graphs have routing number $O(\log n)$ and constant maximum degree, it follows from the subsequent results by Kolman and Scheideler [40] on the routing number that *BoundedGreedy* is an $O(\log n)$-approximation algorithm for MEDP in de Bruijn graphs.

3.4 Expander Graphs

Kleinberg and Rubinfeld [34] considered MEDP in bounded-degree expander graphs. They fix a natural number $\Delta \geq 3$ and a real number $\alpha > 0$ arbitrarily, and then they assume that the maximum degree of the given graph G is at most Δ and that the expansion $\beta(G)$ is at least α.

Theorem 11 (Kleinberg and Rubinfeld, 1996). *For every constant Δ, the bounded-length greedy algorithm with parameter $D = c\Delta \log n$ for a suitable constant c is an $O(\log n \log\log n)$-approximation algorithm for* MEDP *in undirected expander graphs with maximum degree at most Δ.*

To analyze the algorithm, they first show that there exists a feasible routing for a subset $\mathcal{R}'$ of $\mathcal{R}$ that is a partial permutation. Furthermore, $|\mathcal{R}'|$ is at least a constant fraction (more precisely, a $1/(\Delta+1)$-fraction) of the optimal solution for $\mathcal{R}$. This follows, because each vertex is the endpoint of at most Δ requests that are accepted in the optimal solution for $\mathcal{R}$, and so the requests in the optimal solution, viewed as edges between their endpoints, can be partitioned into $\Delta+1$ matchings by Vizing's edge coloring theorem. The largest matching gives the desired partial permutation $\mathcal{R}'$.

Next, they show that there is a set of requests $\mathcal{R}'' \subseteq \mathcal{R}'$ containing at least half of the paths in $\mathcal{R}'$ such that the requests in $\mathcal{R}''$ can be routed along paths of length at most $c \log n$ and at most $c' \log\log n$ paths are routed through the same edge. This result builds on previous work on routing of partial permutations in expanders by Broder, Frieze, and Upfal [12].

To analyze the bounded-length greedy algorithm, the same technique as in Section 2.2 can be used: Whenever the algorithm accepts a request and routes it along a path π, we remove that request and all other requests with paths that intersect π from $\mathcal{R}''$. We see that at most $|\pi| \cdot c' \log\log n + 1 = O(\log n \log\log n)$ requests have to be removed from $\mathcal{R}''$ in this way, thus proving the claimed approximation ratio.

Since expanders have routing number $O(\log n)$, it follows from the work of Kolman and Scheideler [40] that *BoundedGreedy* with parameter $D = O(\log n)$ achieves approximation ratio $O(\log n)$ for expanders, thus improving the result by Kleinberg and Rubinfeld.

Chakrabarti et al. [14] consider so-called Δ-*regular strong expanders* and show that approximation ratio $O(\sqrt{\log n})$ can be achieved for sufficiently large constant Δ.

3.5 Complete Graphs

For undirected or bidirected complete graphs, Erlebach and Vukadinović [22] have shown that there is a solution that uses only paths of length at most two and whose size is at least a constant fraction of the optimal solution. Therefore, the bounded-length greedy algorithm with $D = 2$ and the shortest-path-first greedy algorithm both achieve constant approximation ratio for MEDP in complete graphs. Using the results on the flow number due to Kolman and Scheideler [41], Carmi et al. [13] have shown that the approximation ratio achieved by the shortest-path-first greedy algorithm and the bounded-length greedy algorithm (with $D = 4$) is at most 9 and cannot be better than 3 for undirected complete graphs (which have flow number 1).

Chekuri and Khanna [15] prove that the shortest-path-first greedy algorithm has approximation ratio $O(n/\delta)$ for MEDP in undirected graphs with minimum degree δ. This also implies a constant-factor approximation algorithm for MEDP in complete graphs.

MEDP is $\mathcal{NP}$-hard in undirected and bidirected complete graphs [22], but no inapproximability result is known.

4 MEDP with Pre-determined Paths

In our definition of the MEDP problem, each request specifies only the endpoints s_i and t_i, and if the request is accepted, the path connecting s_i and t_i is determined by the algorithm. Alternatively, one can assume that every request with endpoints s_i and t_i already specifies a pre-determined path π_i connecting s_i and t_i. The path π_i might be determined by a separate routing algorithm or by the customer who submits the request. If request i is accepted, it must be routed along π_i. We denote this variant of MEDP by PreMEDP.

An instance I of PreMEDP with a set of k paths in a graph G can be viewed as a maximum independent set problem in the *conflict graph* of I, i.e., in the graph with one vertex for each of the k paths and with an edge between two vertices if the corresponding paths intersect.

Of course, the problems MEDP and PreMEDP are equivalent for trees. For undirected or bidirected rings, PreMEDP is polynomial, since the conflict graph is a circular-arc graph (or a disjoint union of two circular-arc graphs, in the bidirected case) in this case and the maximum independent set problem is polynomial for circular-arc graphs [30]. For undirected or bidirected trees of rings, PreMEDP was shown $\mathcal{APX}$-complete in [20]. For the approximation ratio achieved by a greedy algorithm based on depth-first search, upper bounds of 4 and 8 were given there for the undirected and bidirected case, respectively. For

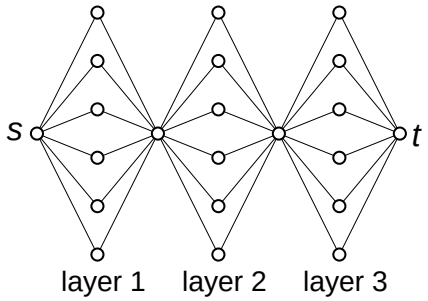

Fig. 9. A series-parallel graph.

arbitrary graphs, it is easy to see that the natural adaptations of the bounded-length greedy algorithm and the shortest-path-first greedy algorithm achieve approximation ratio $O(\sqrt{m})$ for PreMEDP.

To get an inapproximability result for PreMEDP in arbitrary graphs, note that any graph H with n vertices can be encoded as the conflict graph of n paths in a series-parallel graph with $m = O(n^2)$ edges, such as the one shown in Fig. 9: All n paths go from s to t. There are $O(n)$ layers between s and t, and each layer consists of $O(n)$ parallel chains of length two. The edge set of H can be split into $\ell = O(n)$ matchings M_1, M_2, ..., M_ℓ using an edge coloring algorithm. In each layer i, the paths corresponding to the endpoints of the jth edge in M_i both use the jth chain of that layer. The paths corresponding to nodes that are not matched in M_i use one of the other chains of that layer (no two of them use the same chain). Thus, two of the constructed paths intersect if and only there is an edge between the corresponding vertices in H.

The maximum independent set problem in graphs with n vertices cannot be approximated within $n^{1-\varepsilon}$ for any $\varepsilon > 0$ unless $\mathcal{NP} = \text{co-}RP$, as shown by Håstad [32]. Therefore, PreMEDP cannot be approximated within $m^{\frac{1}{2}-\varepsilon}$ for any $\varepsilon > 0$ under the same assumption. Unlike in the case of MEDP, this inapproximability result for PreMEDP applies also to undirected graphs and to graphs with bounded treewidth (since series-parallel graphs have treewidth at most two).

We remark that any graph with n vertices can also be encoded as the conflict graph of n paths in a mesh with $O(n^2)$ vertices, thus yielding the same inapproximability result. This construction was given by Nomikos in a reduction from the vertex coloring problem to path coloring with pre-determined paths [48].

For PreMEDP, one can also take the number of *rejected* paths as the objective value and turn the problem into a minimization problem, as suggested by Blum, Kalai and Kleinberg [6]. Viewed in the conflict graph, the goal is now to compute a minimum vertex cover. Since the minimum vertex cover problem can be approximated within a factor of 2 on arbitrary graphs, this shows that PreMEDP admits a 2-approximation for arbitrary graphs if the goal is to minimize the number of rejected paths.

5 The Unsplittable Flow Problem

The *unsplittable flow problem* (UFP) generalizes MEDP in several aspects. With respect to the given graph $G = (V, E)$, the difference is that every edge e now has a positive capacity $u(e)$. With respect to the set of requests $\mathcal{R}$, the difference is that in addition to specifying its endpoints s_i and t_i, each request i also has a positive *demand* d_i and a positive *profit* r_i. Unless stated otherwise, we assume that the edge capacities, demands, and profits can be arbitrary positive rational numbers. Again, a solution is given by selecting a subset of the given requests and assigning a path from s_i to t_i to each accepted request i. A solution is feasible if the edge capacity constraints are satisfied, i.e., if the sum of the demands of all accepted requests that are routed through an edge e is at most $u(e)$. The objective value of a solution is the sum of the profits of the accepted requests.

For a given instance of UFP, we let $u_{\min}$ and $u_{\max}$ denote the smallest and largest edge capacity, respectively. Similarly, $d_{\min}$ and $d_{\max}$ represent the smallest and largest demand of any request, and $r_{\min}$ and $r_{\max}$ are defined analogously. For a set $\mathcal{S}$ of requests, we define $r(\mathcal{S}) = \sum_{i \in \mathcal{S}} r_i$ and $d(\mathcal{S}) = \sum_{i \in \mathcal{S}} d_i$.

In addition to studying UFP in its full generality, many researchers have considered combinations of the following restrictions.

- The largest demand is at most the smallest edge capacity, i.e., $d_{\max} \leq u_{\min}$. Since many results are known under this assumption, we call UFP with this restriction *classical* UFP. To distinguish the general UFP problem from classical UFP, it is also called *extended* UFP.
- We have $d_{\max} \leq u_{\min}/K$ for some $K > 1$. This variant is called *bounded* UFP.
- All edge capacities are the same. This variant is called *uniform-capacity* UFP (UCUFP).
- All demands are the same.
- All profits are the same.
- The profit of a request is proportional to its demand.

In particular, MEDP is the special case of UFP in which $u(e) = 1$ for all $e \in E$ and $d_i = r_i = 1$ for all requests i. Therefore, all inapproximability results for MEDP apply also to UFP.

Even though UFP appears to be significantly more general than MEDP, an approximation ratio of $O(\sqrt{m})$ can be achieved for classical UFP as well.

5.1 UFP Approximations in Terms of Routing Number and Flow Number

Kolman and Scheideler [40] consider UCUFP in undirected graphs under the assumption that the profit of a request is equal to its demand. They suggest to use the bounded-length greedy algorithm with length parameter $D = 2R$, where R is the routing number of the given graph (cf. Section 2.2). If the algorithm

processes the requests in order of non-increasing demands, they show that approximation ratio $O(\Delta R) = O(\Delta^2 \beta(G)^{-1} \log n)$ is achieved. They also present a randomized algorithm with approximation ratio $O(\Delta\sqrt{LR})$ if the graph has L-bounded routing number R.

In subsequent work, Kolman and Scheideler [41] obtain improved results by considering the *flow number* instead of the routing number. Let $G = (V, E)$ be an undirected graph with arbitrary positive edge capacities. For $v \in V$, let $u(v)$ be the sum of the capacities of the edges incident to v. Let $\Gamma = \sum_{v \in V} u(v)$. Consider a concurrent multicommodity flow problem I with a demand of $d_{v,w} = u(v)u(w)/\Gamma$ between every pair (v, w) of vertices in V. A feasible solution assigns to each pair (v, w) a (splittable) flow of at most $d_{v,w}$ units from v to w such that the total flow through each edge is at most the capacity of the edge. The *flow value* of a feasible solution is the maximum value f, $0 \leq f \leq 1$, such that at least $f \cdot d_{v,w}$ units of flow are routed for each pair (v, w). Finally, the *flow number* $F(G)$ is defined as the minimum, over all feasible solutions S to I, of $\max\{C(S), D(S)\}$, where $D(S)$ is the longest flow path used in S and $C(S)$ is the inverse of the flow value of S [41].

Kolman and Scheideler show that the flow number $F(G)$ of a graph can be computed in polynomial time and, using a result by Leighton and Rao [43], that $F(G) = \Omega(\beta(G)^{-1})$ and $F(G) = O(\Delta'\beta(G)^{-1} \log n)$ always hold, where $\Delta' = \max_{v \in V} u(v)$ and $\beta(G)$ is the expansion of G.

Then they prove that the bounded-length greedy algorithm with length parameter $D = 4F(G)$ achieves approximation ratio $16F(G) + 1 = O(F(G)) = O(\Delta'\beta(G)^{-1} \log n)$ for the classical UFP in undirected graphs if the profit of a request is equal to its demand and if the algorithm processes the requests in order of non-increasing demand values. In terms of Δ instead of Δ', this gives approximation ratio $O(\Delta \frac{u_{\max}}{u_{\min}} \beta(G)^{-1} \log n)$. For bounded UFP with $d_{\max} \leq u_{\min}/K$ for some integer K, they present a modified algorithm with approximation ratio $O(K \cdot (F(G)^{1/K} - 1))$. If $K \geq \log F(G)$, this gives approximation ratio $O(\log F(G))$.

They also use an LP-based algorithm with randomized rounding to show that approximation ratio $O(1)$ can be achieved for classical UFP if $u_{\min}/d_{\max} \geq \gamma \log F(G)$ for a sufficiently large constant γ.

5.2 A Combinatorial Algorithm for Classical UFP

In the following, we present the combinatorial $O(\sqrt{m})$-approximation algorithm for classical UFP due to Azar and Regev [4]. First, the set of requests $\mathcal{R}$ is split into two subsets $\mathcal{R}_1$ and $\mathcal{R}_2$ such that $\mathcal{R}_1$ contains all requests with $d_i \leq u_{\min}/2$ and $\mathcal{R}_2$ all remaining requests. The algorithm computes a solution for each of the two sets separately and outputs the better of the two solutions. This costs at most a factor of 2 in the approximation ratio.

Consider one of the two sets R_q, $q = 1, 2$. The algorithm sorts the requests in order of non-increasing ratio r_i/d_i and processes them in this order. In this way, more "valuable" requests are processed first. For a request i and a path π

```
algorithm ThresholdGreedy(G = (V, E), S, α):
  A ← ∅;
  sort the requests in S in order of non-increasing r_i/d_i;
  for all requests i in S in this order do
      if ∃ path π_i from s_i to t_i in G such that
      F(i, π_i) > α and every edge e ∈ π has at least d_i available capacity then
          A ← A ∪ {(s_i, t_i)};
          reduce the available capacity on all edges of π_i by d_i;
      fi;
  od;
  return A and {π_i | (s_i, t_i) ∈ A};
```

Fig. 10. A threshold-based variant of the greedy algorithm.

from s_i to t_i, define

$$F(i,\pi) = \frac{r_i}{\sum_{e\in\pi} \frac{d_i}{u(e)}}.$$

Thus, $F(i,\pi)$ measures the profit gained relative to the added network load. The main idea of the algorithm is to fix a threshold α and to accept a request i only if there is a path π from s_i to t_i that has sufficient free capacity to accommodate the request and that satisfies

$$F(i,\pi) > \alpha. \tag{6}$$

The best choice for α is not known a priori. Therefore, the algorithm tries all powers of two in a certain range as possible values for α, computes a solution for each of these values, and outputs the best of them. A meaningful range for α is from $\alpha_{\min} = r_{\min}/n$ to $\alpha_{\max} = r_{\max}/(d_{\min}/u_{\max})$, because (6) is satisfied for all paths in case $\alpha \le \alpha_{\min}$ and for no path if $\alpha > \alpha_{\max}$. Thus, the algorithm tries out $\alpha = 2^j$ for $\lfloor \log \alpha_{\min} \rfloor \le j \le \lceil \log \alpha_{\max} \rceil$.

More precisely, the algorithm by Azar and Regev calls the subroutine *ThresholdGreedy* of Fig. 10 with each combination of parameters $\mathcal{S}$ and $\alpha = 2^j$ satisfying $\mathcal{S} = \mathcal{R}_1$ or $\mathcal{S} = \mathcal{R}_2$ and $\lfloor \log \alpha_{\min} \rfloor \le j \le \lceil \log \alpha_{\max} \rceil$. The best of the solutions obtained in this way is output.

To see that the algorithm has a polynomial running-time, note that the subroutine *ThresholdGreedy* is called

$$O\left(\log \frac{\alpha_{\max}}{\alpha_{\min}}\right) = O\left(\log\left(\, n \frac{r_{\max} u_{\max}}{r_{\min} d_{\min}} \,\right)\right)$$

times, so the number of calls is bounded by a polynomial in the size of the input. (Numbers in the input are encoded using a logarithmic number of bits.) Each call of *ThresholdGreedy* can be executed in polynomial time since the existence of a path π_i with enough available capacity and with $F(i,\pi_i) > \alpha$ can be checked using a shortest path algorithm, for the execution of which the weight of an edge e is taken to be $1/u(e)$.

Theorem 12 (Azar and Regev, 2001). *There is an $O(\sqrt{m})$-approximation algorithm for classical* UFP *in arbitrary directed or undirected graphs.*

Proof. Let $\mathcal{Q}$ be the set of requests in an optimal solution, and let π_i^* be the path assigned to request $i \in \mathcal{Q}$ by the optimal solution. Take $q = 1$ or $q = 2$ such that $\mathcal{Q} \cap \mathcal{R}_q$ contains at least half of the total profit of $\mathcal{Q}$. Let $\mathcal{Q}' = \mathcal{Q} \cap \mathcal{R}_q$. Note that $r(\mathcal{Q}') \geq r(\mathcal{Q})/2$.

Let $\alpha' = 2^{j'}$ be the largest value for α that is considered by the algorithm and that satisfies $r(\{i \in \mathcal{Q}' \mid F(i, \pi_i^*) > \alpha'\}) \geq r(\mathcal{Q}')/2$. It is clear that such an α' exists. We will show that *ThresholdGreedy*$(G, \mathcal{R}_q, \alpha')$ produces a solution $\mathcal{A}$ with $r(\mathcal{Q}')/r(\mathcal{A}) = O(\sqrt{m})$. Let π_i be the path assigned to request i in this set $\mathcal{A}$ by the algorithm.

Let $\mathcal{Q}'_{\text{high}} = \{i \in \mathcal{Q}' \mid F(i, \pi_i^*) > \alpha'\}$ and $\mathcal{Q}'_{\text{low}} = \{i \in \mathcal{Q}' \mid F(i, \pi_i^*) \leq 2\alpha'\}$ Note that $\mathcal{Q}'_{\text{high}}$ and $\mathcal{Q}'_{\text{low}}$ are not necessarily disjoint and each of them has total profit at least $r(\mathcal{Q}')/2$. We obtain

$$r(\mathcal{Q}'_{\text{low}}) = \sum_{i \in \mathcal{Q}'_{\text{low}}} F(i, \pi_i^*) \sum_{e \in \pi_i^*} \frac{d_i}{u(e)} \leq 2\alpha' \sum_{i \in \mathcal{Q}'_{\text{low}}} \sum_{e \in \pi_i^*} \frac{d_i}{u(e)} \leq 2\alpha' m \,,$$

since the solution $\mathcal{Q}'$ respects the edge capacities. Thus, we have $r(\mathcal{Q}') \leq 4m\alpha'$ and $r(\mathcal{Q}) \leq 2r(\mathcal{Q}') \leq 8m\alpha'$.

Call an edge e *heavy* if the total demand routed through the edge in the solution $\mathcal{A}$ is at least $u(e)/4$. Then define E_{heavy} to be the set of all heavy edges. We distinguish two cases.

Case 1. E_{heavy} contains at least $\sqrt{m}$ edges. Then we get

$$\begin{aligned} r(\mathcal{A}) &= \sum_{i \in \mathcal{A}} F(i, \pi_i) \sum_{e \in \pi_i} \frac{d_i}{u(e)} \geq \alpha' \sum_{i \in \mathcal{A}} \sum_{e \in \pi_i} \frac{d_i}{u(e)} \\ &\geq \alpha' |E_{\text{heavy}}|/4 \geq \alpha' \sqrt{m}/4. \end{aligned}$$

Thus we get $r(\mathcal{Q}) \leq 8m\alpha' \leq 32\sqrt{m} \cdot r(\mathcal{A})$.

Case 2. E_{heavy} contains less than $\sqrt{m}$ edges. In this case, we compare $r(\mathcal{A})$ with $r(\mathcal{Q}'_{\text{high}})$. Let $\mathcal{Q}'' = \mathcal{Q}'_{\text{high}} \setminus \mathcal{A}$. Since every request i in $\mathcal{Q}''$ is not accepted by the algorithm even though $F(i, \pi_i^*) > \alpha'$, this means that if the algorithm had routed request i along π_i^*, this would have exceeded the capacity of at least one of the edges on π_i^*, say, of the edge e_i.

We claim that e_i is a heavy edge. If $q = 1$, all requests in $\mathcal{R}_q$ have demand at most $u_{\min}/2$, thus the edge e_i can overflow only if it already carries a total demand of more than $u(e_i)/2$ and, consequently, is a heavy edge. If $q = 2$, all demands are between $u_{\min}/2$ and $u_{\min}$. If $u(e_i) \leq 2u_{\min}$, at least one request must already be routed through e_i and thus use at least $u_{\min}/2 \geq u(e_i)/4$ of the capacity of the edge. If $u(e_i) > 2u_{\min}$, a total demand of more than $u(e_i) - d_{\max} \geq u(e_i) - u_{\min} \geq u(e_i)/2$ must already be using the edge e_i. Again, we find that e_i is a heavy edge.

Let $\mathcal{Q}''(p)$ be the set of requests in $\mathcal{Q}''$ that are among the first p requests that are processed by the algorithm, and let $E(p) = \{e_i \mid i \in \mathcal{Q}''(p)\}$. Note

that $E(p) \subseteq E_{\text{heavy}}$ and, therefore, $|E(p)| \leq \sqrt{m}$. Let $\mathcal{A}(p)$ be the requests that are accepted by the algorithm among the first p requests that it processes. Now the idea is to show that the total demand of $\mathcal{A}(p)$ is at least $\Omega(1/\sqrt{m})$ times the total demand of $\mathcal{Q}''(p)$. Since the requests are processed in order of non-increasing ratio r_j/d_j, we will then be able to conclude that $r(\mathcal{A})$ is $\Omega(1/\sqrt{m})$ times $r(\mathcal{Q}'')$.

Since each request in $\mathcal{Q}''(p)$ is routed through an edge of $E(p)$ in the optimal solution, we have $d(\mathcal{Q}''(p)) \leq \sum_{e \in E(p)} u(e)$. Let f be an edge in $E(p)$ with largest capacity. As f is heavy, we have $d(\mathcal{A}(p)) \geq u(f)/4$. So we get

$$d(\mathcal{Q}''(p)) < \sqrt{m} \cdot u(f) \leq 4\sqrt{m} \cdot d(\mathcal{A}(p)).$$

This implies that $r(\mathcal{Q}'') < 4\sqrt{m} \cdot r(\mathcal{A})$, since the requests are processed in order of non-increasing ratio r_i/d_i and the following claim can be verified by elementary calculations: Let $d_1, d_2, \ldots, d_\ell$ be a positive sequence and $b_1, b_2, \ldots, b_\ell$ a non-increasing positive sequence, and let $X, Y \subseteq \{1, \ldots, \ell\}$ and $X(p) = X \cap \{1, \ldots, p\}$ and $Y(p) = Y \cap \{1, \ldots, p\}$. If for every $1 \leq p \leq \ell$ we have $\sum_{j \in X(p)} d_j > \gamma \sum_{j \in Y(p)} d_j$, then

$$\sum_{j \in X} d_j b_j > \gamma \sum_{j \in Y} d_j b_j.$$

By taking $b_j = r_j/d_j$, $X(p) = \mathcal{A}(p)$, $Y(p) = \mathcal{Q}''(p)$, and $\gamma = 1/(4\sqrt{m})$, the above implication follows. From $r(\mathcal{Q}'') < 4\sqrt{m} \cdot r(\mathcal{A})$ and $r(\mathcal{Q}'_{\text{high}}) \leq r(\mathcal{Q}'') + r(\mathcal{A})$, we get $r(\mathcal{Q}'_{\text{high}}) \leq (4\sqrt{m} + 1) r(\mathcal{A})$ and thus $r(\mathcal{Q}) \leq 2r(\mathcal{Q}') \leq 4r(\mathcal{Q}'_{\text{high}}) \leq (16\sqrt{m} + 4) r(\mathcal{A})$.

In both cases, we have shown that $r(\mathcal{Q}) = O(\sqrt{m}) \cdot r(\mathcal{A})$. Thus, the algorithm achieves approximation ratio $O(\sqrt{m})$. □

The running-time of the algorithm leading to Theorem 12 is polynomial in the size of the input, but it depends on the numbers that are part of the input, because *ThresholdGreedy* is called $O(\log\left(n \frac{r_{\max} u_{\max}}{r_{\min} d_{\min}}\right))$ times. Azar and Regev show that it is possible to modify the algorithm so that its running-time is *strongly polynomial*, i.e., bounded by a polynomial function of n, m, and k independent of the edge capacities, profit values, and demand values. To achieve this, first note that the capacity of an edge e with $u(e) > k d_{\max}$ can be reduced to $k d_{\max}$ without affecting the feasible solutions. Next, requests with profit below $r_{\max}/k$ can be discarded, while losing at most a factor of 2 in the approximation ratio. Then, the set $\mathcal{R}_{\text{tiny}}$ of requests with demand at most $u_{\min}/k$ are treated separately: the algorithm computes one solution consisting of all requests in $\mathcal{R}_{\text{tiny}}$ and one solution computed as before for the requests in $\mathcal{R} \setminus \mathcal{R}_{\text{tiny}}$. By outputting the better of the two solutions, again at most a factor of 2 is lost in the approximation ratio. In $\mathcal{R} \setminus \mathcal{R}_{\text{tiny}}$, all requests have demand at least $u_{\min}/k$. Thus, the ratio $r_{\max} u_{\max}/(r_{\min} d_{\min})$ is now bounded from above by $O(k^3)$, so the running-time of the algorithm is strongly polynomial.

Finally, we note here that Chekuri and Khanna [15] have extended their results for MEDP (cf. Section 2.3) to UCUFP: They gave algorithms with

approximation ratio $O(\min\{\sqrt{m}, n^{2/3}\})$ for UCUFP in undirected graphs and $O(\min\{\sqrt{m}, n^{4/5}\})$ for UCUFP in directed graphs, under the assumption that the profit of each request is equal to 1. Their analysis works also if the profit of each request is equal to its demand.

5.3 Combinatorial Algorithms for Extended and Bounded UFP

Azar and Regev [4] present additional results for extended UFP and bounded UFP. For extended UFP, they obtain a combinatorial, strongly polynomial $O(\sqrt{m} \cdot \log(2 + \frac{d_{\max}}{u_{\min}}))$-approximation algorithm. The basic idea of the algorithm is to partition the given requests into $2 + \max\{\log \frac{d_{\max}}{u_{\min}}, 0\}$ classes depending on their demands and to run the algorithm of the previous section on each of the classes separately. In the end, the best of the computed solutions is output.

For bounded UFP, i.e., for the case of $d_{\max} \leq u_{\min}/K$ for some $K \geq 2$, Azar and Regev present a strongly polynomial $O(K \cdot D^{\frac{1}{K}})$-approximation algorithm, where D is the maximum possible length of a path assigned to a request.

For the extended UFP, Kolman and Scheideler [41] propose a variant of the bounded-length greedy algorithm that achieves approximation ratio $O(\sqrt{m})$ under the assumption that the profit of each request is equal to its demand. Kolman [39] generalizes the results due to Chekuri and Khanna [15] for UCUFP and shows that approximation ratio $O(\min\{\sqrt{m}, n^{2/3}\})$ and $O(\min\{\sqrt{m}, n^{4/5}\})$ can be achieved for extended UFP in undirected and directed graphs, respectively, provided that the profit of each request is equal to its demand.

5.4 Inapproximability Results for UFP

In Section 2.4, we have seen that there cannot be an approximation algorithm for MEDP in directed graphs that achieves approximation ratio $O(m^{\frac{1}{2}-\varepsilon})$ for any $\varepsilon > 0$, unless $\mathcal{P} = \mathcal{NP}$. Since UFP is a generalization of MEDP, this inapproximability result applies to UFP as well. For the extended UFP, Azar and Regev present a stronger inapproximability result in [4]. They show that unless $\mathcal{P} = \mathcal{NP}$, no approximation algorithm for the extended UFP can achieve approximation ratio $O(m^{1-\varepsilon})$ for any $\varepsilon > 0$. (In terms of n, the result shows that no algorithm can have approximation ratio $O(n^{1-\varepsilon})$.) For instances of the extended UFP with $d_{\max}/u_{\min} \geq 2$ and any $\varepsilon > 0$, they prove that no approximation algorithm can have ratio better than $\Omega(m^{\frac{1}{2}-\varepsilon}\sqrt{\lfloor \log \frac{d_{\max}}{u_{\min}} \rfloor})$ unless $\mathcal{P} = \mathcal{NP}$. The proofs of these inapproximability results are again based on the $\mathcal{NP}$-completeness of problem 2DirPath and, hence, apply only to directed graphs.

5.5 LP-Based Algorithms for UFP

Prior to the work by Azar and Regev [4], a number of researchers had investigated the possibility of obtaining approximation algorithms for UFP from a

linear programming relaxation. In Section 2.5, we have discussed the natural linear programming relaxation of MEDP. It is not difficult to see that the linear programming formulation of MEDP can be adapted to UFP in a straightforward way. In this context, the requests are often called *commodities* and the LP formulation is called a *fractional multicommodity flow problem*.

In case of extended UFP, it is important to ensure that a request (s_i, t_i) is routed only along paths π on which all edges have capacity at least d_i. This can easily be enforced in the LP formulation.

Throughout this section, we let OPT^* denote the objective value of the optimal solution to the LP-formulation and let OPT denote the integral optimum value. It is clear that $OPT \leq OPT^*$.

Srinivasan [55] presents approximation algorithms for UCUFP that are based on rounding the fractional solution of the LP relaxation. Without loss of generality, one can assume that $u(e) = 1$ for all $e \in E$. Under the assumption that the profit of each request is equal to its demand value, Srinivasan obtains a polynomial-time algorithm that outputs a feasible solution with total profit $\Omega(\max\{(OPT^*)^2/m, OPT^*/\sqrt{m}\})$. The bound $\Omega((OPT^*)^2/m)$ holds also in the case of arbitrary profit values in the interval $[0, 1]$.

Furthermore, Srinivasan considers fractional solutions to the LP with objective value at least $OPT^*/2$ such that each path that carries a positive fraction of a request (i.e., the paths π for which some variable $y_{i,\pi}$ is non-zero) consists of at most ℓ edges. He shows that in this case an $O(\ell)$-approximation is possible for UCUFP with arbitrary profits. Using results from [34], $\ell = O(\Delta^2 \beta^{-2} \log^3 n)$ can always be achieved for UCUFP. Thus, for graphs in which Δ is bounded by a constant and β^{-1} is polylogarithmic, there is an approximation algorithm for UCUFP with polylogarithmic (in n) approximation ratio. This gives polylogarithmic approximation ratio for the butterfly and related hypercubic networks, which have $\Delta = O(1)$ and $\beta = \Theta(1/\log n)$. Srinivasan also shows that approximation ratio $O(\ell^{1/\varepsilon - 1})$ can be achieved for UCUFP in the case where $d_{\max} \leq 1 - \varepsilon$ for some constant $\varepsilon \geq \frac{1}{2}$. This implies that for graphs like the butterfly, approximation ratio $O(1)$ can be achieved for UCUFP provided that $d_{\max} = O(1/\log\log n)$.

Kolliopoulos and Stein [37] obtain results concerning column-restricted packing integer programs and use them to derive LP-based algorithms for MEDP and classical UFP. For MEDP with arbitrary profit values, their algorithm computes a solution with total profit $\Omega(OPT^*/\sqrt{m})$ provided that the number k of requests is $O(m)$. For classical UFP with profit values in the interval $[0, 1]$, their solutions have total profit $\Omega(OPT^*/(\sqrt{m}\log m))$, $\Omega((OPT^*)^2/(m\log^3 m))$ and $\Omega(OPT^*/\ell)$, where ℓ is defined as above.

Baveja and Srinivasan generalize the results of [55] to classical UFP in [5]. They show that it is possible to compute in polynomial time a solution with total profit at least $\Omega(\min\{OPT^*, (OPT^*)^2/m\})$ and $\Omega(OPT^*/\sqrt{m})$. The approximation algorithms with ratio $O(\ell)$ and $O(\ell^{1/\varepsilon - 1})$ of [55] are also generalized to classical UFP and bounded UFP, respectively.

Guruswami et al. [31] consider LP-based algorithms that apply the randomized rounding technique [50] in a more direct way. They assume that all edge capacities and demand values are integral and that $d_{\max}$ is bounded by a polynomial in m. Their algorithms achieve approximation ratio $O(\sqrt{m}\log^{\frac{3}{2}} m)$ for extended UFP and $O(\sqrt{m\log m}\log\log m)$ for classical UFP. They also show that if $u_{\min}/d_{\max} \geq c\log m$ for a suitably large constant c, then UFP can be approximated within a constant factor by standard randomized rounding (cf. Section 2.5).

Chakrabarti et al. [14] give LP-based algorithms to obtain approximation ratio $O(\Delta\beta(G)^{-1}\log^2 n)$ for UFP in undirected graphs and $O(\Delta\beta(G)^{-1}\log n)$ for UCUFP in undirected graphs. They also show that the approximation ratios improve to $O((\Delta\beta(G)^{-1}\log^2 n)^{1/K})$ for UFP and $O((\Delta\beta(G)^{-1}\log n)^{1/K})$ for UCUFP in the bounded case with $d_{\max} \leq u_{\min}/K$. Chakrabarti et al. [14] and Chekuri et al. [16] use LP-based algorithms also to obtain constant-factor approximation algorithms for classical UFP in chains, rings, and trees.

6 Further Results for Related Problems

In this survey we have focused on approximation algorithms for MEDP and UFP. Numerous results for closely related problems can be found in the literature. In this section we briefly mention some of them.

First, maximum path coloring (MAXPC) is the variant of MEDP where the input specifies an additional parameter $W \geq 1$ and the goal is to accept and route a subset of the requests such that the resulting paths can be partitioned into at most W sets of edge-disjoint paths. This problem is motivated by all-optical networks with wavelength-division multiplexing, because a network with W wavelengths can establish connections for W sets of edge-disjoint paths simultaneously. By a general reduction [17, 2], a ρ-approximation algorithm for MEDP can be converted into an approximation algorithm with ratio at most $1/(1-\mathrm{e}^{-1/\rho}) \leq \rho + 1$ for MAXPC. In some cases, better approximation ratios for MAXPC have been obtained using more direct approaches, for example by Nomikos et al. for MAXPC in undirected and bidirected rings [49].

Another problem related to MEDP is *path coloring*, where *all* given requests must be routed and assigned colors such that requests receive different colors if they share an edge. The goal is to minimize the number of colors used. Results for path coloring are surveyed in a different chapter of this book.

A variant of MEDP, called the bounded-length edge-disjoint paths problem, specifies a bound L on the lengths of the paths that the algorithm may assign to the requests. An $O(\sqrt{m})$-approximation for this problem is presented in [31]. That paper deals also with the integral-splittable flow problem, i.e., with the variant of UFP where the requests need not be routed along single paths, but can be split integrally among several paths (i.e., each demand is an integer and can be split among several paths in an integral way).

Instead of considering the maximization version of the edge-disjoint paths problem, one can also consider the question of how many terminal pairs in a

graph can *always* be connected along edge-disjoint paths, no matter how the terminal pairs are chosen. For r-regular expander graphs, all sets of up to $\kappa = \Omega(n/\log n)$ pairs of vertices can be connected along edge-disjoint paths (provided no vertex appears as an endpoint of more than a constant number of requests), both in the undirected case [25] and in the directed case [9]. Since random graphs have good expansion properties with high probability, similar results could also be proved for random graphs of sufficiently high degree [11] and for random regular graphs [26].

While we have seen that UFP cannot be approximated within $O(m^{\frac{1}{2}-\varepsilon})$ unless $\mathcal{P} = \mathcal{NP}$, it has been shown that the single-source version of the problem admits constant-factor approximation algorithms [18, 54]. In the single-source version, the vertex s_i of all requests (s_i, t_i) is the same.

Finally, we emphasize that we have mainly considered approximation algorithms for MEDP and UFP that have full knowledge about the input, i.e., that are off-line algorithms. In applications such as call admission control, it is meaningful to consider the on-line version of the problem, where the requests arrive over time and the algorithm must accept or reject each request without knowledge of future requests. The greedy algorithm and the bounded-length greedy algorithm that we discussed as approximation algorithms can in fact be applied as on-line algorithms, since they can process the requests in an arbitrary order. For surveys of known results on on-line MEDP and UFP, we refer the reader to Leonardi [44] and Borodin and El-Yaniv [10, Chapter 13]. More recent results can be found in Azar and Regev [4] and Kolman and Scheideler [41].

Acknowledgements

The author would like to thank Petr Kolman and an anonymous referee for numerous comments that helped to improve the presentation.

References

1. Y. Aumann and Y. Rabani. Improved bounds for all optical routing. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'95)*, pages 567–576, 1995.
2. B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi, and A. Rosén. On-line competitive algorithms for call admission in optical networks. *Algorithmica*, 31(1):29–43, 2001.
3. B. Awerbuch, R. Gawlick, T. Leighton, and Y. Rabani. On-line admission control and circuit routing for high performance computing and communication. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS'94)*, pages 412–423, 1994.
4. Y. Azar and O. Regev. Strongly polynomial algorithms for the unsplittable flow problem. In *Proceedings of the 8th Integer Programming and Combinatorial Optimization Conference (IPCO)*, LNCS 2081, pages 15–29, 2001.
5. A. Baveja and A. Srinivasan. Approximation algorithms for disjoint paths and related routing and packing problems. *Mathematics of Operations Research*, 25(2):255–280, May 2000.

6. A. Blum, A. Kalai, and J. Kleinberg. Admission control to minimize rejections. In *Proceedings of the 7th International Workshop on Algorithms and Data Structures (WADS 2001)*, LNCS 2125, pages 155–164, 2001.
7. H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.
8. H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209:1–45, 1998.
9. T. Bohman and A. Frieze. Arc-disjoint paths in expander digraphs. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS'01)*, pages 558–567, 2001.
10. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
11. A. Z. Broder, A. M. Frieze, S. Suen, and E. Upfal. Optimal construction of edge-disjoint paths in random graphs. *SIAM J. Comput.*, 28(2):541–573, April 1999.
12. A. Z. Broder, A. M. Frieze, and E. Upfal. Existence and construction of edge-disjoint paths on expander graphs. *SIAM J. Comput.*, 23(5):976–989, October 1994.
13. P. Carmi, T. Erlebach, and Y. Okamoto. Greedy edge-disjoint paths in complete graphs. In *Proceedings of the 29th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'03)*, LNCS 2880, pages 143–155, 2003.
14. A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation algorithms for the unsplittable flow problem. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX'02)*, LNCS 2462, pages 51–66, 2002.
15. C. Chekuri and S. Khanna. Edge disjoint paths revisited. In *Proceedings of the 14th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'03)*, pages 628–637, 2003.
16. C. Chekuri, M. Mydlarz, and F. Shepherd. Multicommodity demand flow in a tree. In *Proceedings of the 30th International Colloquium on Automata, Languages, and Programming (ICALP'03)*, LNCS 2719, pages 410–425, 2003.
17. G. Cornuejols, M. L. Fisher, and G. L. Nemhauser. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management Science*, 23(8):789–810, April 1977.
18. Y. Dinitz, N. Garg, and M. X. Goemans. On the single source unsplittable flow problem. *Combinatorica*, pages 1–25, 1999.
19. T. Erlebach. *Scheduling Connections in Fast Networks*. PhD thesis, Technische Universität München, 1999.
20. T. Erlebach. Approximation algorithms and complexity results for path problems in trees of rings. In *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS 2001)*, LNCS 2136, pages 351–362, 2001. See also TIK-Report 109, Computer Engineering and Networks Laboratory (TIK), ETH Zürich, June 2001.
21. T. Erlebach and K. Jansen. The maximum edge-disjoint paths problem in bidirected trees. *SIAM Journal on Discrete Mathematics*, 14(3):326–355, 2001.
22. T. Erlebach and D. Vukadinović. New results for path problems in generalized stars, complete graphs, and brick wall graphs. In *Proceedings of the 13th International Symposium on Fundamentals of Computation Theory (FCT 2001)*, LNCS 2138, pages 483–494, 2001.
23. S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980.

24. A. Frank. Packing paths, circuits, and cuts – a survey. In B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, editors, *Paths, Flows, and VLSI-Layout*, pages 47–100. Springer-Verlag, Berlin, 1990.
25. A. M. Frieze. Edge-disjoint paths in expander graphs. *SIAM J. Comput.*, 30:1790–1801, 2001.
26. A. M. Frieze and L. Zhao. Optimal construction of edge-disjoint paths in random regular graphs. *Combinatorics, Probability and Computing*, 9:241–264, 2000.
27. M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York-San Francisco, 1979.
28. N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.
29. Q.-P. Gu and H. Tamaki. Routing a permutation in the hypercube by two sets of edge disjoint paths. *Journal of Parallel and Distributed Computing*, 44(2):147–152, 1997.
30. U. I. Gupta, D. T. Lee, and J. Y.-T. Leung. Efficient algorithms for interval graphs and circular-arc graphs. *Networks*, 12:459–467, 1982.
31. V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC'99)*, pages 19–28, 1999.
32. J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS'96)*, pages 627–636, 1996.
33. J. Kleinberg. *Approximation algorithms for disjoint paths problems*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1996.
34. J. Kleinberg and R. Rubinfeld. Short paths in expander graphs. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS'96)*, pages 86–95, 1996.
35. J. Kleinberg and É. Tardos. Approximations for the disjoint paths problem in high-diameter planar networks. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC'95)*, pages 26–35, 1995.
36. J. Kleinberg and É. Tardos. Disjoint paths in densely embedded graphs. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, pages 52–61, 1995.
37. S. G. Kolliopoulos and C. Stein. Approximating disjoint-path problems using greedy algorithms and packing integer programs. In *Proceedings of the 6th Integer Programming and Combinatorial Optimization Conference (IPCO VI)*, LNCS 1412, pages 153–168, 1998.
38. P. Kolman. Short disjoint paths on hypercubic graphs. Technical Report 2000-481, KAM-DIMATIA Series, Charles University, Prague, 2000.
39. P. Kolman. A note on the greedy algorithm for the unsplittable flow problem. Manuscript, December 2002.
40. P. Kolman and C. Scheideler. Simple on-line algorithms for the maximum disjoint paths problem. In *Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'01)*, pages 38–47, 2001.
41. P. Kolman and C. Scheideler. Improved bounds for the unsplittable flow problem. In *Proceedings of the 13th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'02)*, pages 184–193, 2002.

42. M. E. Kramer and J. van Leeuwen. The complexity of wire routing and finding the minimum area layouts for arbitrary VLSI circuits. In F. P. Preparata, editor, *Advances in Computing Research; VLSI Theory*, volume 2, pages 129–146. JAI Press Inc., Greenwich, CT-London, 1984.
43. T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, November 1999.
44. S. Leonardi. On-line network routing. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, LNCS 1442. Springer-Verlag, Berlin, 1998.
45. B. Ma and L. Wang. On the inapproximability of disjoint paths and minimum Steiner forest with bandwidth constraints. *Journal of Computer and System Sciences*, 60:1–12, 2000.
46. M. Middendorf and F. Pfeiffer. On the complexity of the disjoint paths problem. *Combinatorica*, 13(1):97–107, 1993.
47. R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
48. C. Nomikos. Approximability of the path coloring problem. In *ASHCOMP Workshop*, Udine, 1996.
49. C. Nomikos, A. Pagourtzis, and S. Zachos. Minimizing request blocking in all-optical rings. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003)*, 2003.
50. P. Raghavan and C. D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.
51. N. Robertson and P. Seymour. Graph Minors XIII: The disjoint paths problem. *Journal of Combinatorial Theory Series B*, 63:65–110, 1995.
52. C. Scheideler. *Universal Routing Strategies for Interconnection Networks*. LNCS 1390. Springer-Verlag, 1998.
53. A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, Chichester, 1986.
54. M. Skutella. Approximating the single source unsplittable min-cost flow problem. *Mathematical Programming Series B*, 91(3):493–514, 2002.
55. A. Srinivasan. Improved approximations for edge-disjoint paths, unsplittable flow, and related routing problems. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS'97)*, pages 416–425, 1997.
56. K. Varadarajan and G. Venkataraman. Graph decomposition and a greedy algorithm for edge-disjoint paths. In *Proceedings of the 15th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'04)*, pages 379–380, 2004.
57. J. Vygen. Disjoint paths. Technical Report 94816, Research Institute for Discrete Mathematics, University of Bonn, February 1994.
58. P.-J. Wan and L. Liu. Maximal throughput in wavelength-routed optical networks. In *Multichannel Optical Networks: Theory and Practice*, volume 46 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 15–26. AMS, 1998.
59. X. Zhou, S. Tamura, and T. Nishizeki. Finding edge-disjoint paths in partial k-trees. *Algorithmica*, 26:3–30, 2000.