Improved Bounds for Online Routing and Packing Via a Primal-Dual Approach

Niv Buchbinder
Computer Science Department
Technion, Haifa, Israel
E-mail: nivb@cs.technion.ac.il

Joseph (Seffi) Naor*
Microsoft Research
Redmond, WA 98052
E-mail: naor@cs.technion.ac.il

Abstract

In this work we study a wide range of online and offline routing and packing problems with various objectives. We provide a unified approach, based on a clean primal-dual method, for the design of online algorithms for these problems, as well as improved bounds on the competitive factor. In particular, our analysis uses weak duality rather than a tailor made (i.e., problem specific) potential function. We demonstrate our ideas and results in the context of routing problems.

Using our primal-dual approach, we develop a new generic *online routing algorithm that outperforms previous algorithms suggested earlier by Azar et al. [5, 4]. We then show the applicability of our generic algorithm to various models and provide improved algorithms for achieving coordinate-wise competitiveness, maximizing throughput, and minimizing maximum load. In particular, we improve the results obtained by Goel et al. [13] by an $O(\log n)$ factor for the problem of achieving coordinate-wise competitiveness, and by an $O(\log \log n)$ factor for the problem of maximizing the throughput. For some of the settings we also prove improved lower bounds. We believe our results further our understanding of the applicability of the primal-dual method to online algorithms, and we are confident that the method will prove useful to other online scenarios.*

Finally, we revisit the notions of coordinate-wise and prefix competitiveness in an offline setting. We design the first polynomial time algorithm that computes an almost optimal coordinate-wise routing for several routing models. We also revisit previously studied routing models [16, 11] and prove tight lower and upper bounds of $\Theta(\log n)$ on prefix competitiveness for these models.

1 Introduction

In this work we study a wide range of allocation problems with various objectives. In the most general setting, we are given a set of clients and facilities, where each facility has bounded capacity, and thus it can only be allocated to a limited number of clients. In general, a client may require service from a subset of the facilities (and not just one facility). Thus, clients specify, as part of the input, subsets of the facility set that can provide service to them. An allocation algorithm has to decide which clients to serve, and for each served client it needs to choose a single service option. We consider mostly online settings, in which clients arrive one-by-one in an adversarial mode, and allocation decisions cannot be reversed. We also consider certain offline settings in which all requests are known in advance. Interesting special cases of this very general framework are e.g., the well studied routing and tree packing problems in a network. The facilities in these problems correspond to the links of the network. In the routing problem each client specifies source and destination vertices, bandwidth demand, and a set of feasible paths connecting the source to the destination. In the case of tree packing, each client specifies a set of terminals, demands, and a set of feasible Steiner trees that span the set of terminals. To keep the discussion simple and clear, we demonstrate our ideas and results in the setting of routing problems, and later on explain the necessary modifications for handling other packing problems.

1.1 Routing

Routing and call admission problems in various models have been studied extensively in both offline and online settings. Consider a network modelled by a graph $G = (V, E)$ ($|V| = n$, $|E| = m$), which can be either directed or undirected. The edges in the graph have capacities, denoted by $u : E \to \mathbb{N}$, which provide an upper bound on the sum of the demands of the routes that can be packed into an edge. The set of routing requests is $\mathbb{R}$ and, for simplicity, each request $r_i \in \mathbb{R}$ is associated with a bandwidth demand of one unit[1] between a source vertex s_i and a target vertex t_i. In

*On leave from the Computer Science Dept., Technion, Haifa, Israel.

[1] Our results can be extended, with obvious limitations, to handle scenarios in which requests have different bandwidth demands. We elaborate on this later on.

order to serve a request r_i we should allocate bandwidth for the request on paths that connect the source vertex s_i to the target vertex t_i. There are several common ways by which this can be done. The setting in which each request should be served via a single path is referred to as *unsplittable routing*. A less restrictive setting in which each request can be served via multiple routes is called *splittable routing*. We associate each request with a set of allowed paths (routes) $\mathbb{P}(r_i)$, capturing the *fixed routes* model, in which requests can only be served via a *unique* given path, as a special case. Let $b(r_i)$ be the sum of all bandwidth allocations assigned to request r_i on all paths $P \in \mathbb{P}(r_i)$. The *total bandwidth* of a routing solution is the total bandwidth allocated to all the requests. A *feasible* routing solution is an allocation of bandwidth to requests that does not violate any of the edge capacities. When the routing solution is infeasible, the *load* on an edge is the total bandwidth allocated to it divided by its capacity. The load of a routing solution is the maximum load taken over all edges.

An important parameter that is used in our analysis, as well as in previous analysis, is U, which is defined to be the minimum value by which the capacities in the network need to be multiplied so as to obtain a feasible splittable solution that routes all requests. When routes are fixed, U reduces to the maximum, taken over all edges, of the number of routes that pass through an edge divided by its capacity.

Routing models differ with respect to the issue of whether requests have to be fully served or not. *All-or-nothing* routing means that a request has to be allocated a total bandwidth (splittable or unsplittable) of one unit [10]. Other models relax this requirement and allow the routing algorithm to allocate requests less than one unit of bandwidth. The work of [13] introduced another model, primarily of interest in online settings, in which a routing algorithm allocates a weight $w(r_i, P)$ to request r_i on path P. Eventually, the bandwidth given to request r_i on path P is its "fair" share with respect to its weight. That is, the bandwidth allocated to request r_i on path P, $b(r_i, P) = \min\{1, \min_{e \in P}\{u(e)\frac{w(r_i,P)}{w(e)}\}\}$, where $w(e)$ is the total weight allocated to all paths (of all requests) using edge e.

Routing algorithms are designed to achieve several natural goals. One goal is to maximize the *utility* of the network which is the total bandwidth allocated to all requests. In a somewhat dual setting, the routing algorithm is not allowed to reject any of the requests, in which case the goal is to minimize the maximum load. Another important routing goal is *fairness*. An accepted notion of fairness is *max-min* fairness [7, 14]. To define a fair routing solution, we consider the bandwidth allocation to the requests ($b(r_i)$ to request r_i) as a vector in which the entries (allocations) are sorted from small to large. This vector is called a *bandwidth vector*. A max-min fair routing solution is then an allocation of bandwidth to requests which defines a lexicographically maximal bandwidth vector. An intuitive way of viewing a max-min fair solution is that one cannot increase the bandwidth allocation to a request r_i without decreasing the bandwidth allocated to requests that have received at most the bandwidth given to r_i.

A more general fairness measure was suggested by Kleinberg et al. [15]. A routing solution is called γ_c-*coordinate-wise competitive* if for every i, the ith coordinate of the bandwidth vector is at least $1/\gamma_c$ times the ith coordinate in any feasible routing solution. The beauty of this definition is that a γ_c-coordinate-wise competitive routing approximates *all* possible routings. In particular, it approximates the max-min fair routing, as well as the routing solution that maximizes the total bandwidth allocated, achieving, in some sense, a solution which is the "best of all worlds". Kleinberg et al. [15] also studied a slightly weaker notion of *prefix competitiveness*. A routing solution is γ_P-prefix competitive if for every i, the sum of the first i coordinates in the bandwidth vector is at least $1/\gamma_P$ times the sum of the first i coordinates in any feasible routing solution. It is not hard to see that a γ_c-coordinate-wise competitive routing is also γ_c-prefix competitive routing. The converse, however, is not necessarily true.

Previous Work: Routing algorithms have been studied extensively. We thus mention only results that are directly relevant to our current work. In [4, 5] two different (but similar in spirit) online routing algorithms were suggested. The objective in [5] is maximizing the total throughput, while the algorithm in [4] minimizes the load. These algorithms, as well as the ideas behind them, are closely related to our results and are further discussed in Section 2.1. In [8] an online routing algorithm, based on a primal-dual approach, was suggested. The algorithm achieves a splittable routing which is $O(\log n)$-competitive. The algorithm draws on ideas from previous work on covering problems [1, 2].

The elegant notion of *max-min fairness* was considered in many settings [7, 14]. The general framework of prefix and coordinate-wise competitiveness was suggested in [15]. More properties of these measures (in the offline case) were studied later on in [16, 11]. Goel et al. [13] studied the problem of achieving coordinate-wise competitiveness online. They designed an algorithm which is $O(\frac{1}{\epsilon}\log^2 n(\log U)^{1+\epsilon})$-coordinate-wise competitive for any $\epsilon > 0$. In a relaxed setting where the algorithm is allowed to assign weights instead of allocating bandwidth directly, [13] designed an algorithm which is $O(\log^2 n \log U)$-coordinate-wise competitive. Routing and load balancing are closely related problems [6]. The online load balancing problem in the 1-∞ model was studied from a fairness perspective in [12]. In this model each job has weight 1 and it can be assigned to a (given) subset of the machines. They proved that the greedy allocation yields an assignment which is $O(\log n)$-prefix competitive, where n is the number of jobs. This result was recently improved by [9] who showed that the greedy allocation is, in fact, $O(\log m)$-prefix competitive, where m is the number of machines.

1.2 Results

We provide a unified approach to a wide range of online routing and packing problems with several objectives, as well as improved bounds on the competitive factor. Our unified approach is based on a clean primal-dual method for designing online routing algorithms. The primal-dual method is one of the fundamental design methodologies in the areas of approximation algorithms and combinatorial optimization. Recently, Buchbinder and Naor [8] have further extended the primal-dual method and have shown its applicability to the design and analysis of online algorithms. We use the primal-dual method here for both the routing algorithms as well as for their analysis. Moreover, we observe that known online routing algorithms can actually be viewed and analyzed within the primal-dual framework, thus leading to both simpler and more general analysis. The analysis uses, for example, weak duality rather than a tailor made (i.e., problem specific) potential function. We believe our results further our understanding of the primal-dual method for online algorithms and we are confident that the method will prove useful in other online scenarios as well.

Two parameters are of particular interest in routing problems. The first one is the amount of *bandwidth* that the algorithm routes with respect to an optimal routing, and the second one is the maximum *load* on the edges. A (c_1, c_2)-competitive routing algorithm routes at least $1/c_1$ of the maximum possible bandwidth, while guaranteeing that the load on each edge is at most c_2. With this notation in mind we re-examine previously suggested algorithms. The first algorithm we consider was suggested in [5] for maximizing the throughput. Using our notation this algorithm is $(O(\log n), 1)$-competitive, provided that the minimal edge capacity is at least $\log n$. We show a simple primal-dual construction and analysis of this algorithm. A simple change in the algorithm also leads to a construction of an $(O(1), O(\log n))$-competitive algorithm, that does not need the restriction on the edges' capacities. This algorithm is actually, a bicriteria competitive algorithm that routes a constant fraction of the optimal number of requests while incurring a load of $O(\log n)$. Next, we reanalyze the properties of the algorithm in [4] using our primal-dual view. This algorithm guarantees that if it is possible to route offline **all** requests without exceeding the capacities, then the algorithm also routes all requests incurring a load of at most $O(\log n)$. The algorithm has no guarantee if only part of the requests can be routed offline.

Our main result is the construction, using a primal-dual approach, of a *generic* general purpose routing algorithm that outperforms the algorithms of [5] and [4]. Specifically, our algorithm is $(1, O(\log n))$-competitive, thus obtaining a uni-criteria competitive algorithm, as well as satisfying the guarantee of the algorithm in [4]. A simple example shows that such a result is optimal for an online algorithm. The construction of an algorithm with such performance guarantees turns out to be non-trivial[2]. Our generic algorithm generates an unsplittable all-or-nothing routing; however, to allow the use of the algorithm in a wide variety of routing models, its performance is compared to a splittable optimal routing which is allowed to allocate to each request (total) bandwidth in the interval $[0, 1]$. The stronger performance guarantees of our generic algorithm facilitate the design of online routing solutions for several models and objectives, yielding improved bounds. In what follows we list the main applications of the generic algorithm for which we obtain improved bounds. We note that there are more applications for which the known bounds are already tight, yet our generic algorithm can be used to derive the same bounds, e.g., [5, 4, 8]. For lack of space we defer the details to the full version. We find it very elegant that a single unified routing algorithm can be used as a "black box" to derive algorithms for so many settings and objectives.

In addition, we observe that our generic online algorithm does not use the fact that edges in each feasible route indeed induce an actual path in the graph. This property is very useful for extending our results to general packing models, i.e., to the case where general service options $S \in \mathbb{S}(r_i)$ are given for each request r_i. For example, each request may consist of a set of terminals, and the "service sets" are all Steiner trees that span the terminals. In this case, the set of feasible solutions is given implicitly, and the algorithm only requires that there exists an *oracle* that gets as input a weight function on the edges and outputs a minimum weight solution. In case the oracle can only compute a β-approximate solution, the algorithm outputs a $(1, O(\beta \log n))$-competitive solution. Dealing with approximate oracles requires only a small change in the algorithm. For more details see Section 2.

Application 1: coordinate-wise competitiveness. The first objective we study is achieving online a coordinate-wise competitive solution. Two settings were previously studied with respect to this measure. First, a setting with fixed routes for each request [16], and a second setting in which each request is allowed to be served in a splittable

[2] We can easily transform a (c_1, c_2)-competitive algorithm to a $(c_1 \cdot c_2, 1)$-competitive algorithm by scaling down all allocated bandwidth. However, obtaining a $(1, c_1 \cdot c_2)$-competitive factor is problematic, since requests should be allocated bandwidth of at most 1.

Models		Coordinate-wise competitiveness	
Routes	Allocation	Lower Bound	Upper Bound (Unsplittable)
Splittable	Bandwidth	$\boldsymbol{\Omega(\log n \log U)}$ + $\Omega(\log U \log\log U)$ [13]	$\boldsymbol{O(\frac{1}{\epsilon} \log n \log U (\log\log U)^{1+\epsilon})}$
	Weight	$\Omega(\log n + \log U)$ [13, 16]	$\boldsymbol{O(\log n \log U)}$
Fixed	Bandwidth	$\boldsymbol{\Omega(\log n + \log n \log \frac{U}{\log n})}$	$\boldsymbol{O(\frac{1}{\epsilon} \log n + \frac{1}{\epsilon} \log n \log \frac{U}{\log n} (\log\log \frac{U}{\log n})^{1+\epsilon})}$
	Weight	$\Omega(\log U)$ [16]	$\boldsymbol{O(\log n + \log n \log \frac{U}{\log n})}$

Figure 1. Upper/Lower bounds on online coordinate-wise competitiveness. Our results are in bold.

fashion [11]. Each setting is then divided into two online models which were considered in [13]: In the first model, denoted as a *bandwidth* model, the algorithm should allocate each request bandwidth directly in the interval $[0, 1]$. In the second model, the *weight* model, the algorithm is allowed to give weights to the requests, instead of directly allocating bandwidth. Figure 1 summarizes the best known results on online coordinate-wise competitiveness. For each model we state the best known upper and lower bounds on achieving coordinate-wise competitiveness online. We remark that our algorithm results in an unsplittable routing, thus all the upper bounds in Figure 1 for the splittable case are actually stronger. Our results appear in Figure 1 in bold and are summarized in the following:

- An $O(\log n)$ improvement in the competitiveness over the algorithms in [13] (for both models).
- A new, almost tight, lower bound of $\Omega(\log n \log U)$ for the bandwidth model.
- Slightly better competitive algorithms and suitable improved lower bounds when routes are fixed.

Application 2: maximizing throughput. The second objective we study is maximizing the total throughput. In [13] a model where requests should be routed in an unsplittable way and get bandwidth in the interval $[0, 1]$ is considered. The goal is to maximize the total throughput of the network. Goel at al. [13] designed an $O(\log n \log\log n)$-competitive algorithm for the problem and proved an $\Omega(\log n)$ lower bound on the competitive ratio of any deterministic algorithm for the problem. We design a simple algorithm based on our generic algorithm that tightens the competitive ratio to $O(\log n)$. We also extend their lower bound to the fixed routes model. We remark that the algorithm in [5] can also be used in this setting. However, the algorithm in [5] requires the minimal capacity of an edge to be at least $\log n$, while our algorithm does not have this requirement.

Application 3: minimizing maximum load. The third objective we study is minimizing the maximum load. In this setting all requests should be fully routed, and the objective is to minimize the maximum load. In [4] an optimal $O(\log n)$-competitive algorithm for the problem was suggested. Their algorithm is based on an underlying algorithm with weaker guarantees than our generic algorithm (See section 2.1). We show a simple alternative way of applying our generic algorithm that achieves the same (optimal) competitive ratio. Even though we do not achieve a better competitive ratio, we claim that using our algorithm achieves other desired goals, since our algorithm guarantees that the load on the edges is more evenly spread among the edges. For more details see Section 3.4.

Offline prefix and coordinate-wise competitiveness. Finally, we revisit offline questions that arise in the context of prefix and coordinate-wise competitiveness [16]. In general, there is no reason that a single 1-competitive (prefix or coordinate-wise) solution exists with respect to **all** solutions. Thus, the first issue is determining the best possible competitiveness in each of the models under consideration. A separate issue is whether such a solution can be efficiently computed. Two routing settings were previously studied with respect to these issues: a fixed routes model [16] and a splittable routing model [11]. Each model can be further divided into a *restricted* setting in which no two requests can share both their source and target vertices (and thus there are at most n^2 requests), and an *unrestricted* setting in which two different requests may share both source and target vertices. We close all the remaining open questions regarding these two models. Specifically, we prove that the prefix competitiveness is $\Theta(\log n)$ for both models, while the coordinate-wise competitiveness is $\Theta(\log U)$. In general, the value U can be as large as the number of requests, which is typically much larger then the size of the network. Thus, this result shows a separation between the notion of coordinate-wise competitiveness that is $\Theta(\log U)$ and the notion of prefix competitiveness that is $\Theta(\log n)$, and thus depends only on the size of the network and not on the number of requests. Finally, we show how to efficiently compute an $O(\log U + \frac{\log n}{\log\log n})$-coordinate-wise competitive routing with respect to all splittable routings. Our contributions are the following:

- Construction of a general polynomial time algorithm that computes an (almost optimal) unsplittable

Primal		Dual	
Minimize:	$\sum_{e\in E} u(e)x(e) + \sum_{r_i} Z(r_i)$	Maximize:	$\sum_{r_i}\sum_{P\in\mathbb{P}(r_i)} f(r_i,P)$
Subject to:		Subject to:	
$\forall r_i \in \mathbb{R}, P \in \mathbb{P}(r_i)$:	$\sum_{e\in P} x(e) + Z(r_i) \geq 1$	$\forall r_i \in \mathbb{R}$:	$\sum_{P\in\mathbb{P}(r_i)} f(r_i,P) \leq 1$
		$\forall e \in E$:	$\sum_{r_i\in\mathbb{R},P\in\mathbb{P}(r_i)\mid e\in P}\sum_{r_i} f(r_i,P) \leq u(e)$

Figure 2. A primal-dual pair for the splittable routing problem.

$O(\log U + \frac{\log n}{\log\log n})$-coordinate-wise competitive solution with respect to all splittable routings. For this problem only an existential result was known [16].

- Tight upper and lower bounds of $\Theta(\log n)$ on prefix competitiveness for both the fixed routes model and the splittable model. This result improves upon the previous $O(\log U)$ upper bound and the $\Omega(\frac{\log n}{\log\log n})$ lower bound for the fixed routes model obtained by Kumar and Kleinberg [16][3]. For the splittable model this result extends the proof in [11] to an unrestricted setting.
- Proof of the existence of an (optimal) $O(\log U)$-coordinate-wise competitive routing solution for the splittable routing model.

2 A Generic Online Routing Algorithm

In this section we first inspect previously suggested routing algorithms and show how they fit into our primal-dual framework. We then design a generic online routing algorithm which is based on the primal-dual approach and which outperforms these algorithms. All the algorithms we describe get requests r_i in an online fashion that consist of a source vertex s_i, a target vertex t_i, and a set of feasible routes $\mathbb{P}(r_i)$ that can be used to serve each request. The set of possible routes, $\mathbb{P}(r_i)$, can be given either explicitly or implicitly. When the routes are given implicitly, we require that there exists an oracle that gets a weight function $w : E \to \mathbb{R}^+$ and returns the shortest path $P \in \mathbb{P}(r_i)$ with respect to the weight function.

An optimal splittable routing can be formulated as a linear program. The formulation appears in Figure 2 as the dual (maximum) linear program with the objective of maximizing the total throughput. We have a variable $f(r_i, P)$ for each request r_i and path $P \in \mathbb{P}(r_i)$ indicating the amount of bandwidth allocated to request r_i on path P. The first set of constraints guarantees that the total bandwidth allocated to each request is at most 1. The second set of constraints guarantees that the edge capacities are not violated. The corresponding primal program has a variable $Z(r_i)$ for each request and a variable $x(e)$ for each edge e. The constraints of the primal program stipulate that for any request r_i and any path $P \in \mathbb{P}(r_i)$, the sum of the variables $x(e)$, taken over all edges e on the path P, plus the variable $Z(r_i)$, is at least 1.

An online routing algorithm generates a solution to the dual program in Figure 2. Since our goal is to compare our performance to an optimal splittable routing, all the algorithms we describe maintain at all times a solution to the primal program that upper bounds the value of our current dual solution. It is interesting to note that the online request arrival can actually be viewed as gradually revealing both the dual and the corresponding primal programs.

2.1 Previous Online Routing Algorithms

Our primal-dual approach can be used to describe and analyze several previously suggested routing algorithms. These algorithm were previously described and analyzed using problem specific potential function. The first relevant algorithm ($\mathbf{AAP}$) was suggested in [5] for maximizing the throughput. The algorithm can be described very simply using our primal-dual approach.

> $\mathbf{AAP}$: Initially: $x(e) \leftarrow 0$.
> When new request $r_i = (s_i, t_i, \mathbb{P}(r_i))$ arrives:
>
> 1. If there exists a path $P(r_i) \in \mathbb{P}(r_i)$ of length < 1 with respect to $x(e)$:
>   (a) Route the request on **any** path $P(r_i) \in \mathbb{P}(r_i)$ with length < 1.
>   (b) $Z(r_i) \leftarrow 1$.
>   (c) For each edge e in $P(r_i)$:
>
> $$x(e) \leftarrow x(e)\exp\left(\frac{\ln(1+n)}{u(e)}\right) + \frac{1}{n}\left[\exp\left(\frac{\ln(1+n)}{u(e)}\right) - 1\right]$$

Lemma 2.1. *Algorithm* $\mathbf{AAP}$ *is* $\left(O(u(\min)\cdot\left[\exp\left(\frac{\ln(1+n)}{u(\min)}\right)-1\right], 1\right)$*-competitive with respect to all splittable routing solutions. If* $u(\min) \geq \log n$ *then the algorithm is* $(O(\log n), 1)$*-competitive.*

Proof sketch. The main observation is that when a request

[3] In fact, our result shows a bound of $\min\{O(\log n), O(\log U)\}$. We observe that the proof in [16] actually yields $\gamma_P = \Omega(\frac{\log n}{\log\log n})$, rather than $\Omega(\frac{\log U}{\log\log U})$, since the family of graphs used for the lower bound proof has the property that n, the number of vertices, grows as U increases.

Primal		Dual	
Minimize:	$\sum_{e\in E} x(e) + \sum_{r_i} Z(r_i)$	Maximize:	$\sum_{r_i} \sum_{P\in\mathbb{P}(r_i)} f(r_i,P)$
Subject to:		Subject to:	
$\forall r_i \in \mathbb{R}, P \in \mathbb{P}(r_i)$:	$\sum_{e\in P} \frac{x(e)}{u(e)} + Z(r_i) \geq 1$	$\forall r_i \in \mathbb{R}$:	$\sum_{P\in\mathbb{P}(r_i)} f(r_i,P) \leq 1$
		$\forall e \in E$:	$\frac{1}{u(e)} \sum_{r_i\in\mathbb{R}, P\in\mathbb{P}(r_i)\mid e\in P} f(r_i,P) \leq 1$

Figure 3. A primal-dual pair for the problem in [4] .

r_i is routed, the increase of the primal cost is at most $2\left(u(\min)\cdot\left[\exp\left(\frac{\ln(1+n)}{u(\min)}\right)-1\right]\right)+1$. This follows since $Z(r_i) \leftarrow 1$, and since for edges on the path $P(r_i)$ satisfy $\sum_{e\in P(r_i)} x(e) \leq 1$. Whenever the algorithm decides to reject a request, the change in the primal cost is zero. The second observation is that the algorithm maintains a feasible primal solution at all times. This follows since $Z(r_i) \leftarrow 1$ for each request with shortest path less than 1. Each time a request is routed, the dual profit is 1. Thus, by weak duality the algorithm routes at least the fraction claimed in the lemma of the maximal possible requests.

It remains to prove that the algorithm routes at most $u(e)$ requests on each edge e. To this end, observe that for each edge e, the value $x(e)$ is the sum of a geometric sequence with initial value $\frac{1}{n}\left[\exp\left(\frac{\ln(1+n)}{u(e)}\right)-1\right]$ and multiplier $\exp\left(\frac{\ln(1+n)}{u(e)}\right)$. Thus, after $u(e)$ requests are routed through edge e, the value $x(e)$ is:

$$\begin{aligned} x(e) &= \frac{1}{n}\cdot\left(\exp\left(\frac{\ln(1+n)}{u(e)}\right)-1\right)\cdot\frac{\exp\left(\frac{u(e)\ln(1+n)}{u(e)}\right)-1}{\exp\left(\frac{\ln(1+n)}{u(e)}\right)-1} \\ &= \frac{1}{n}\cdot(1+n-1) \geq 1. \end{aligned}$$

Since the algorithm never routes requests on edges for which $x(e) \geq 1$, we are done. ☐

Remark 2.2. *It is interesting to note that a small change in the update rule of $x(e)$ to:*

$$x(e) \leftarrow x(e)\left(1+\frac{1}{u(e)}\right)+\frac{1}{n\cdot u(e)}$$

yields an algorithm which is $(O(1), O(\log n))$-competitive. This algorithm violates the capacity constraints by $O(\log n)$, but does not require that the minimal capacity is at least $\log n$.

The second relevant algorithm (**AAFPW**) was suggested in [4] for the problem of minimizing the maximum load. The main idea used for the analysis of this algorithm is looking at a normalized version of the system that appears in Figure 2. The primal-dual normalized pair is shown in Figure 3. With this normalized form the "heart" of this online algorithm can be described as follows:

AAFPW: Initially: $x(e) \leftarrow \frac{1}{2m}$ (m is the number of edges). When a new request r_i arrives:

1. Let $P(r_i) \in \mathbb{P}(r_i)$ be the shortest path with respect to $\frac{x(e)}{u(e)}$, and let α be the length of $P(r_i)$.
2. If $\alpha \geq 1$ or there is an edge such that $x(e) > 2$ return "FAIL", else:
   (a) Route the request on $P(r_i)$.
   (b) For each edge e in $P(r_i)$: $x(e) \leftarrow x(e)(1+\frac{1}{2u(e)})$.
   (c) $Z(r_i) \leftarrow 1-\alpha$.

Lemma 2.3. *If there exists a* **feasible** *splittable routing solution that routes* **all** *requests, then* **AAFPW** *routes* **all** *the requests and the maximum load is $O(\log n)$.*

Proof. The second part of the lemma is easier. We return "FAIL" when $x(e) > 2$. Thus, if p is the number of requests we route on an edge e, then: $\frac{1}{2m}(1+\frac{1}{2u(e)})^{p-1} \leq 2$. Therefore, the number of requests that the algorithm routes on edge e is at most $u(e)\cdot O(\log n)$. To prove the first part of the lemma, note first that the total initial cost of the primal solution is $1/2$. In any iteration in which the algorithm routes a request, the total change in the cost of the primal solution is:

$$1-\alpha+\sum_{e\in P(r_i)}\frac{x(e)}{2u(e)} = 1-\alpha+\alpha/2 = 1-\alpha/2.$$

The generated primal solution is always feasible since we update $Z(r_i) \leftarrow 1-\alpha$. Assume now that the algorithm returns "FAIL" after routing $N' < N$ requests. There are two possibilities:

First case: $\alpha \geq 1$. This means that the primal solution of the previous phase is feasible without any additional updates. Thus, we have a primal solution with cost $1/2 + N' < N$. This contradicts the fact that we have a feasible dual solution with profit N.

Second case: For some edge e, $x(e) > 2$. Since $x(e)$ is multiplied by $1+\frac{1}{2u(e)}$ in each iteration, it means that in more than $u(e)$ iterations we increased the weight of $x(e)$ while $x(e) \geq 1$. In those iterations, $\alpha \geq \frac{x(e)}{u(e)} \geq \frac{1}{u(e)}$. Thus, after so many iterations, and when setting $Z(r_i) = 1$ for the last request (that was rejected), we obtain a feasible primal solution with cost strictly less than:

$$1/2 + (N' - u(e)) + u(e)(1 - \frac{1}{2u(e)}) + 1 = N,$$

where the first term ($= 1/2$) is the initial cost, the term $N' - u(e)$ upper bounds the cost of the iterations in which $x(e) \leq 1$, the term $u(e)(1 - \frac{1}{2u(e)})$ upper bounds the cost of the $u(e)$ iterations in which $x(e) > 1$, and the last term ($= 1$) is the cost of the last request. This, again, contradicts the fact that we have a feasible dual solution with value N, thus completing the proof . □

Finally, we remark that the load balancing algorithm in [4] can also be described using our primal dual framework. We defer this description to the full version.

2.2 A New Generic Routing Algorithm

In this section we design a generic online routing algorithm which is based on the primal-dual approach. The algorithm then generates in an online fashion an unsplittable all-or-nothing routing which is $(1, O(\log n))$-competitive with respect to all splittable routings. The main point here is that the algorithm actually routes at least the optimal number of requests. Algorithm **AAFPW** guarantees the routing of an optimal number of requests only when there is a feasible solution that routes all the requests. Otherwise, there is no guarantee on the number of requests that this algorithm routes. This weaker guarantee of **AAFPW** suffices for the design of an algorithm for minimizing the maximum load. However, the weaker guarantee does not seem to suffice when we want to maximize the total throughput, or design a coordinate-wise competitive algorithm, as is done in Sections 3.1, 3.2, and 3.3. The **AAP** algorithm is guaranteed to route at least $1/(c \log n)$ of the optimal number of requests without violating the capacities (when the capacities are at least $\log n$). A slight change of **AAP** leads to an algorithm that is $(O(1), O(\log n))$-competitive without any restriction on the capacities (See remark 2.2). This may seem like an insignificant difference with respect to our new guarantees, however, these weaker guarantees become a real obstacle when trying to use this algorithm for minimizing the maximum load, or for achieving coordinate-wise competitiveness, as done in Sections 3.1, 3.2, and 3.4. We also remark that the methods in [8] allow us to obtain the same guarantees as the **AAP** algorithm by applying online derandomization methods.

To achieve our stronger bounds it is not enough to maintain a single primal solution, leading us to maintain simultaneously several primal solutions that will be used throughout to make clever routing decisions.

The algorithm decomposes the graph $G = (V, E)$ into graphs $G_0, G_1, \ldots, G_k$. For each j, the vertices of G_j are the same as in G. The edges of G_j are all edges in G having capacity at least m^j. The capacity of each edge in the jth copy, G_j, is then set to $u(e, j) \leftarrow \min\{u(e), m^{j+2}\}$. Let G_k be the last copy of the graph which is non-empty (i.e. the maximum capacity in G, $u(\max) \leq m^k$). The algorithm maintains a primal solution in each copy of the graph. We denote by $x(e, j)$ and $Z(r_i, j)$ the primal variables corresponding to the jth copy. Let $u(\min, j)$ be the minimal edge capacity in the jth copy (which is at least m^j).

Route: Initially, $\forall j$: $x(e, j) \leftarrow \frac{u(\min,j)}{m \cdot u(e,j)}$.
When new request $r_i = (s_i, t_i, \mathbb{P}(r_i))$ arrives:

1. Consider all copies of the graph from G_k to G_0. In each copy G_j:
   - (a) Let $P(r_i, j) \in \mathbb{P}(r_i, j)$ be the shortest path with respect to $x(e, j)$ and let α be the length of $P(r_i, j)$.
   - (b) If $\alpha < 1$:
     - i. Route the request on $P(r_i, j)$.
     - ii. For each edge e in $P(r_i, j)$: $x(e, j) \leftarrow x(e, j)(1 + \frac{1}{u(e,j)})$.
     - iii. $Z(r_i, j) \leftarrow 1 - \alpha$.
   - (c) Else ($\alpha > 1$):
     - i. If the total bandwidth routed in this step in G_j is less than $u(\min, j)$, and the current request can be routed in G_j, route the request in an arbitrary feasible path $P \in \mathbb{P}(r_i, j)$.
   - (d) If the request is routed - Finish.
2. Reject requests that were rejected from all copies.

The analysis of **Route** is done via the following claims. The proofs of Lemma 2.4 and Theorem 2.5 are omitted.

Lemma 2.4. *Let N_j be the total number of requests that are introduced to the jth copy. Let M be the maximum total bandwidth of any feasible splittable routing in G_j (out of N_j). Then,* **Route** *accepts at least M requests in G_j, and the load on each edge in G_j is $O(\log n)$.*

Theorem 2.5. *Algorithm* **Route** *is $(1, O(\log n))$-competitive with respect to all splittable routing solutions.*

Further Extensions: If the algorithm is only given an approximate oracle (as is the case when packing objects other than paths) that outputs for a request r_i a set $S \in \mathbb{S}(r_i)$ that is at most β times the minimum $S \in \mathbb{S}(r_i)$ with respect to the given weights, the competitive factor of the algorithm becomes $(1, O(\beta \log n))$. This requires the following change in the update method in Steps c(ii) and c(iii): $x(e, j) \leftarrow x(e, j)(1 + \frac{1}{\beta u(e,j)})$ and $Z(r_i, j) \leftarrow \frac{1-\alpha}{\beta}$. A second extension applies to the case where request r_i has an arbitrary demand d_i. Our algorithm generates an unsplittable routing. In case the demands are arbitrary, it might not be possible to compare our algorithm to the best splittable routing. However, it is still possible to compare the performance to the best unsplittable routing, or to an optimal splittable routing that only uses paths with minimal capacity of at least d_i. Other than that, only a few more ideas, and an extension of the proofs are needed. A third extension we consider is adding different weights (profits) to the requests. This enables, for instance, to model a weighted max-min fair solution. For this variant it is possible to design an $(1, O(\log \left[n \frac{p(\max)}{p(\min)}\right]))$-competitive algorithm, where $p(\max)$ and $p(\min)$ are the maximum and minimum profits. A simple example shows that the additional additive factor of $\log(p(\max)/p(\min))$ is unavoidable (See [8]).

3 Applications of the Generic Algorithm

In this section we show the applicability of our generic algorithm to different settings having various objective functions. We note that there are more applications for which the known bounds are already tight, yet our generic algorithm can be used to derive the same bounds, e.g., [5, 4, 8]. For lack of space we defer the details to the full version. In Section 3.1 we design an algorithm for achieving coordinate-wise competitiveness. In Section 3.2 we design a similar algorithm with better competitive factors for a setting that allows the allocation of weights instead of bandwidths. In Section 3.3 we design an $O(\log n)$-competitive algorithm for maximizing the throughput for the setting studied in [13] and extend the lower bounds for the problem. In Section 3.4 we design a scheme for applying our generic algorithm for minimizing the maximum load. We remark that achieving the bounds in Section 3.1 and Section 3.2 for the fixed routes model requires a small modification to the generic algorithm that ensures that when the algorithm rejects a request there exists an edge with load at least $\log n$. The proofs are omitted.

3.1 Achieving Coordinate-wise Competitiveness - Assigning Bandwidth

Here we design an almost optimal online algorithm for achieving a coordinate-wise routing solution [13]. In this setting the algorithm should output an unsplittable routing and assign bandwidth to each request. We design an $O(\frac{1}{\epsilon}\log n \log U (\log\log U)^{1+\epsilon})$ competitive algorithm for any $\epsilon > 0$ and prove an almost matching lower bound of $\Omega(\log n \log U + \log U \log\log U)$ even when splittable routing is allowed. This result improves both upper and lower bounds for the problem, each by a logarithmic factor [13]. For the fixed routes model the competitive ratio of our algorithm slightly improves to $O(\frac{1}{\epsilon}\log n \log \frac{U}{\log n}(\log\log \frac{U}{\log n})^{1+\epsilon})$ for any $\epsilon > 0$. We show an almost matching lower bound of $\Omega(\log n \log \frac{U}{\log n} + \log n)$. The algorithm considers copies of the graph referred to as levels. In levels $\ell = 0, 1, 2, \ldots$ we multiply all edge capacities by 2^ℓ.

Algorithm: When a request $r_i = (s_i, t_i, \mathbb{P}(r_i))$ arrives:

1. Run algorithm $\mathbf{Route}$ on levels $\ell = 0, 1, 2, \ldots$ in an increasing order.
2. Route the request in the lowest level ℓ in which $\mathbf{Route}$ accepts the request.
3. Assign bandwidth of $\frac{\epsilon}{c \log n 2^\ell (1+\ell)(\mathbb{H}(1+\ell))^{1+\epsilon}}$ to the request, where c is a constant and $\mathbb{H}(\cdot)$ is the harmonic number.

Theorem 3.1. *The algorithm is $O(\frac{1}{\epsilon}\log n \log U (\log\log U)^{1+\epsilon})$-coordinate-wise competitive. For the fixed routes model the algorithm is $O(\frac{1}{\epsilon}\log n \log \frac{U}{\log n}(\log\log \frac{U}{\log n})^{1+\epsilon} + \frac{\log n}{\epsilon})$-coordinate-wise competitive.*

Lower bounds: In [13] a lower bound of (approximately) $\Omega(\log n + \log U \log\log U)$ was proved. We improve the lower bound and prove an almost matching lower bound for the problem.

Lemma 3.2. *Any deterministic online algorithm (splittable or unsplittable) is $\Omega(\log n \log U)$-coordinate-wise competitive for the general setting, and $\Omega(\log n \log \frac{U}{\log n} + \log n)$-competitive when routes are fixed.*

3.2 Achieving Coordinate-wise Competitiveness - Assigning Weights

In this section we design an algorithm for achieving a coordinate-wise competitive solution in a relaxed setting studied in [13] in which the algorithm is allowed to assign weights instead of directly allocating bandwidths. For this model we design an $O(\log n \log U)$-coordinate-wise competitive algorithm improving the competitive ratio obtained in [13] by a factor of $O(\log n)$. When routes are fixed the competitive ratio of our algorithm slightly improves to $O(\log n \log \frac{U}{\log n} + \log n)$. The algorithm considers copies of the graph referred to as levels. In levels $\ell = 0, 1, 2, \ldots$ the edge capacities are multiplied by 2^ℓ.

Algorithm: When request $r_i = (s_i, t_i, \mathbb{P}(r_i))$ arrives:

1. Run algorithm $\mathbf{Route}$ in levels $\ell = 0, 1, 2, \ldots$ in an increasing order.
2. Route the request in the lowest level ℓ in which $\mathbf{Route}$ accepts the request and assign the request weight $w(r_i, P) = \frac{1}{2^\ell}$.

Theorem 3.3. *The algorithm is $O(\log n \log U)$-coordinate-wise competitive for the general setting and is $O(\log n \log \frac{U}{\log n} + \log n)$-coordinate-wise competitive for the fixed routes model.*

3.3 Maximizing the Throughput

In [13] Goel et al. considered a routing model in which the algorithm should output a **feasible** unsplittable routing by assigning bandwidth directly to each request. The objective is to maximize the total throughput. This model differs from the model considered in [5], since here there is no additional requirement that the routing is an all-or-nothing routing. Goel et al. [13] designed for this model an algorithm that is $O(\log n \log\log n)$ competitive. In this section we improve their result and design a simple optimal $O(\log n)$-competitive algorithm for maximizing the throughput in this model. We remark that the algorithm in [5] can also be used in this setting. However, the algorithm in [5] requires the minimal capacity to be at least $\log n$, while our algorithm does not have this requirement. Using $\mathbf{Route}$, our algorithm is very simple:

Algorithm: When a request $r_i = (s_i, t_i, \mathbb{P}(r_i))$ arrives:

1. Run algorithm $\mathbf{Route}$ on the graph. Accept all requests that were accepted by $\mathbf{Route}$ on the path $\mathbf{Route}$ chose for the request.
2. Give each accepted request bandwidth $\frac{1}{c \log n}$, where $c > 0$ is some constant.

Lemma 3.4. *The algorithm is $O(\log n)$-competitive with respect to an optimal splittable routing and it does not violate the capacities of the edges in the graph.*

Lower bounds: In [13], an $\Omega(\log n)$ lower bound was proved even in the case where the algorithm can assign weights instead of bandwidth. We observe that the bound of $\Omega(\log n)$ holds for bandwidth allocation even for the fixed routes model.

Lemma 3.5. *There is an $\Omega(\log n)$ lower bound on the competitive ratio of any deterministic online algorithm that assigns bandwidth directly even for the fixed routes model.*

3.4 Minimizing the Maximum Load

In this section we design an alternative $O(\log n)$-competitive algorithm to the problem of minimizing the maximum load [4]. In this setting the algorithm is not allowed to reject requests and the objective is to minimize the load. Although we do not improve over the performance of the algorithm in [4], we claim that using our algorithm may still be beneficial, since our algorithm guarantees that the load on the edges is more evenly spread among the edges. This is due to the fact that the algorithm routes the maximum number of requests with respect to any value of load. The algorithm in [4] only attempts to guess the load in which all requests can be routed. When the guess is less than the minimum load there is no guarantee on the number of requests that will be routed. The algorithm considers copies of the graph referred to as levels. In levels $\ell = 0, 1, 2, \ldots$ all the edge capacities are multiplied by 2^ℓ.

Algorithm: When a request $r_i = (s_i, t_i, \mathbb{P}(r_i))$ arrives:

1. Run algorithm $\mathbf{Route}$ on level $\ell = 0, 1, 2, \ldots$ in an increasing order.
2. Route the request on the lowest level ℓ in which $\mathbf{Route}$ accepts the request. Use the path $\mathbf{Route}$ chosen for the request in this level.

Lemma 3.6. *The algorithm is $O(\log n)$-competitive with respect to the optimal splittable routing solution.*

4 Offline Cases - Revisiting

In this section we study the notions of coordinate-wise competitiveness and prefix competitiveness and their offline computational complexity. We first prove a tight $\Theta(\log n)$ bounds on the prefix competitiveness for both the fixed routes model and the splittable routing model, where n is the number of vertices in the network. This improves both upper and lower bounds of [16] and generalizes the result in [11]. Our second contribution is proving the existence of an $O(\log U)$-coordinate-wise competitive routing for any splittable routing instance. We then design an algorithm for computing efficiently an unsplittable $O(\log U + \frac{\log n}{\log\log n})$-coordinate-wise competitive solution with respect to all splittable routings. For this problem only an existential result was previously known [16]. The proofs are omitted.

4.1 Prefix Competitiveness

In this section we prove upper and lower bounds of $\Theta(\log n)$ on prefix competitiveness in the offline case, thus improving upon the results in [16]. The bounds hold for the splittable model as well as the fixed routes model. The lower bound holds even with the restriction that no two requests can share the same source and target.

Lemma 4.1. *For any routing instance in both the fixed routes model and the splittable model there exists a solution which is $O(\log n)$-prefix competitive.*

Lemma 4.2. *There exists a routing instance, where no two requests have the same source and target vertices, such that any routing solution has prefix competitiveness $\Omega(\log n)$.*

4.2 Coordinate-wise Competitiveness

In this section we sketch the ideas of a non-polynomial algorithm that computes an $O(\log U)$-coordinate-wise competitive routing solution in the splittable model. We then show how to transform it to a polynomial time algorithm losing only a small additive penalty of $O(\frac{\log n}{\log\log n})$. Our proposed algorithm works in phases: In the ith phase the capacities in the graph are multiplied by 2^i. We then compute an optimal feasible all-or-nothing splittable routing for the set of requests that were not routed by any previous phase. We give each request that was routed in the ith phase bandwidth of $1/2^i$. This solution violates the capacity constraints. Yet, we claim that this routing solution is 2-coordinate-wise competitive. Suppose that the jth (poorest) coordinate in some solution is b. Then, at least $N-j+1$ requests in this solution receive bandwidth of at least b. This means that at least $N-j+1$ requests can be routed in an all-or-nothing fashion when multiplying the capacities by $1/b$. By the properties of the algorithm at least $N-j+1$ requests are, therefore, routed until phase $i \leq 2/b$. All those requests get bandwidth at least $b/2$ and thus we are done. By the definition of U, all the requests are routed until phase $i \leq 2U$. Therefore, the load on each edge is at most $\log(2U)$. Thus, dividing each bandwidth allocation by $O(\log U)$, we get a feasible splittable solution that is $O(\log U)$-coordinate-wise competitive.

The non-polynomial step in our algorithm is the computation of an optimal all-or-nothing splittable routing. We thus remove the all-or-nothing requirement, substituting this step by a computation of an optimal feasible splittable routing for the set of requests that were not routed by any previous phase, using, e.g., linear programming. Let $\Lambda^*(i)$ be the maximum bandwidth that was routed in the ith phase. This value upper bounds the optimal all-or-nothing splittable routing. Using standard rounding techniques it is possible to obtain from this splittable routing an unsplittable all-or-nothing routing with value $\Lambda^*(i)$, and load of $\max\{c, O(\frac{\log n}{u(\min)\log(\log n/u(\min))})\}$, where $u(\min)$ is the current minimum capacity, and $c > 1$ is a constant. We note that our performance requirements are slightly different from previous requirements [18], however, we can still obtain a rounding, even deterministically, using the method of conditional expectations [17, 3]. The rest of the algorithm remains the same. Summing up the loads in all phases, we get that the load on each edge is $O(\log U + \frac{\log n}{\log\log n})$. By dividing each allocated bandwidth by this factor, we get a feasible solution with the desired competitiveness.

Lemma 4.3. *For any splittable routing instance there exists an $O(\log U)$-coordinate-wise competitive routing solution. There is a polynomial time algorithm that computes an unsplittable $O(\log U + \frac{\log n}{\log\log n})$-coordinate-wise competitive routing with respect to splittable routing solutions.*

Acknowledgement

We would like to thank the anonymous referees for a very thorough reading of the paper and helpful comments.

References

[1] N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder, and J. Naor. The online set cover problem. In *Proceedings of the 35th annual ACM Symposium on the Theory of Computation*, pages 100–105, 2003.

[2] N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder, and J. Naor. A general approach to online network optimization problems. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 100–105, 2004.

[3] N. Alon and J. H. Spencer. *The probabilistic method*. Wiley, New York, 2 edition, 2000.

[4] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *J. ACM*, 44(3):486–504, 1997.

[5] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive online routing. In *Proc. of 34th FOCS*, pages 32–40, 1993.

[6] Y. Azar. On-line load balancing. *Online Algorithms - The State of the Art, Springer*, (8):178–195, 1998.

[7] D. Bertsekas and R. Gallager. *Data networks*. Prentice-Hall, Inc., 1987.

[8] N. Buchbinder and J. Naor. Online primal-dual algorithms for covering and packing problems. In *13th Annual European Symposium on Algorithms - ESA 2005*, 2005.

[9] N. Buchbinder and J. Naor. Fair online load balancing. In *18th ACM Symp. on Parallelism in Algorithms and Architectures*, 2006.

[10] C. Chekuri, S. Khanna, and F. B. Shepherd. The all-or-nothing multicommodity flow problem. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 156–165, 2004.

[11] A. Goel and A. Meyerson. Simultaneous optimization via approximate majorization for concave profits or convex costs. 2005.

[12] A. Goel, A. Meyerson, and S. A. Plotkin. Approximate majorization and fair online load balancing. In *Symposium on Discrete Algorithms*, pages 384–390, 2001.

[13] A. Goel, A. Meyerson, and S. A. Plotkin. Combining fairness with throughput: Online routing with multiple objectives. *J. Comput. Syst. Sci.*, 63(1):62–79, 2001.

[14] J. Jaffe. Bottleneck flow control. *IEEE Transactions on Communications*, 29(7):954–962, 1981.

[15] J. Kleinberg, E. Tardos, and Y. Rabani. Fairness in routing and load balancing. In *FOCS '99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, page 568, 1999.

[16] A. Kumar and J. M. Kleinberg. Fairness measures for resource allocation. In *IEEE Symposium on Foundations of Computer Science*, pages 75–85, 2000.

[17] P. Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *J. Comput. Syst. Sci.*, 37(2):130–143, 1988.

[18] P. Raghavan and C. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.