Computing Nash Equilibria for Scheduling on Restricted Parallel Links[*]

Martin Gairing[†] Thomas Lücking[†] Marios Mavronicolas[‡] Burkhard Monien[†]

ABSTRACT

We consider the problem of routing n users on m parallel links, under the restriction that each user may only be routed on a link from a certain set of *allowed links* for the user. Thus, the problem is equivalent to the correspondingly restricted problem of assigning n jobs to m parallel machines. In a *pure* Nash equilibrium, no user may improve its own *individual cost* (delay) by unilaterally switching to another link from its set of allowed links. As our main result, we introduce a polynomial time algorithm to compute from any given assignment a pure Nash equilibrium with non-increased makespan. The algorithm gradually changes a given assignment by pushing unsplittable user traffics through a network that is defined by the users and the links. Here, we use ideas from blocking flows. Furthermore, we use similar techniques as in the generic PREFLOW-PUSH algorithm to approximate a schedule with minimum makespan, gaining an improved approximation factor of $2 - \frac{1}{w_1}$ for identical links, where w_1 is the largest user traffic. We extend this result to related links, gaining an approximation factor of 2. Our approximation algorithms run in polynomial time. We close with *tight* upper bounds on the coordination ratio for pure Nash equilibria.

Categories and Subject Descriptors

F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General

[*]This work has been partially supported by the IST Program of the European Union under contract numbers IST-1999-14186 (ALCOM-FT) and IST-2001-33116 (FLAGS). Parts of the work were done, while the last author was visiting the University of Cyprus.

[†]Faculty of Computer Science, Electrical Engineering and Mathematics, University of Paderborn, Fürstenallee 11, 33102 Paderborn, Germany. Email: {gairing,luck,bm}@uni-paderborn.de

[‡]Department of Computer Science, University of Cyprus, P. O. Box 20537, Nicosia CY-1678, Cyprus. Email: mavronic@ucy.ac.cy

General Terms

Algorithms, Theory

Keywords

Selfish routing, machine scheduling, Nash equilibria, unsplittable flow, approximation algorithms

1. INTRODUCTION

Motivation and Framework: The concept of Nash equilibria has become an important mathematical tool for analyzing the behavior of selfish users in non-cooperative systems [22]. Many algorithms have been developed to compute a Nash equilibrium in a general game (see [20] for an overview). Although the theorem of Nash [21] guarantees the existence of a Nash equilibrium for any finite game, the complexity of computing a Nash equilibrium in general games is open even if only two users are involved.

In this work, we continue the study of Nash equilibria for the selfish routing problem introduced by Koutsoupias and Papadimitriou [17], where n selfish users wish to route their traffics $w_1, \ldots, w_n$ through a shared *network* consisting of two *nodes* and m parallel *links* with capacities $c_1, \ldots, c_m$. This is widely known as the *KP-model*. Here, each user chooses a probability distribution over all links as its strategy, trying to minimize its expected latency without cooperating with other users or adhering to a global objective function, the so-called *social cost* (or makespan). A stable state in which no user has an incentive to unilaterally change its strategy is called *Nash equilibrium*. The Nash equilibrium is *pure* if each user chooses a single link.

In contrast to general non-cooperative games, there always exists a *pure* Nash equilibrium with minimum social cost in the KP-model, and there exist polynomial algorithms to compute pure Nash equilibria on *related* links. However, the most general setting of *unrelated* links seems to be intractable right now. We study a special case of the model of unrelated links: each user $i \in [n]$ is only allowed to ship its traffic along a subset $A_i \subseteq [m]$ of *allowed links*.

Contribution and Significance: We first consider the problem of computing an assignment approximating the minimum social cost. This problem is a special case of the general single source unsplittable flow problem. Recently, many approaches have been made to compute a solution for the single source unsplittable flow problem. They are either based on rounding techniques [13, 14] or they directly compute an unsplittable flow from a splittable flow [3]. Applying these techniques yields an approximation factor of 2.

In this work, we choose a simpler approach. We use similar techniques as in the generic PREFLOW-PUSH algorithm[10], only allowing to push traffics without splitting them. This yields an approximation factor of $2 - \frac{1}{w_1}$ if the links are identical, using $O(mA\log(W))$ time, where $A = \sum_{i\in[n]} |A_i|$, and $W = \sum_{i\in[n]} w_i, w_i \in \mathbb{N}$ is the total traffic. This improves on the best known approximation factor for the job scheduling problem on unrelated parallel machines. We extend the algorithm to related links, gaining an approximation factor of 2. Both algorithms are combinatorial, they do not rely on linear programming.

For the model of identical links, we present as our main result, an algorithm which computes a pure Nash equilibrium for users with restricted assignments from any given (not necessarily equilibrium) assignment without increasing the social cost. This technique is called *Nashification*. The algorithm uses techniques from BLOCKING-FLOWS and needs $O(nmA(\log W + m^2))$ time. To the best of our knowledge this is the first polynomial-time algorithm computing a Nash equilibrium in this model. Computing a Nash equilibrium in the model of related links for users with restricted assignments remains open. Of course, this also holds for unrelated links.

We close by showing that for the model of arbitrary users on identical links, the coordination ratio for pure Nash equilibria is $\Theta(\frac{\log(m)}{\log(\log(m))})$. The coordination ratio is the ratio between the worst and the best social cost over all Nash equilibria. This holds even for the model of identical users on identical links. For arbitrary users on related links, the bound $m-1$ is shown to be tight.

Related Work and Comparison: The scheduling problem with restricted assignments is a special case of the problem of scheduling jobs on unrelated parallel machines, originally considered by Horowitz and Sahni [12].

For the problem of scheduling jobs on unrelated machines, Horowitz and Sahni [12] present a (non-polynomial) dynamic programming algorithm to compute a schedule with minimum social cost; they also present a FPTAS to approximate an optimum schedule for the case of *constant m*. For the general case, Lenstra et al. [18] prove that unless $\mathcal{P} = \mathcal{NP}$, there is no polynomial-time approximation algorithm for the optimum schedule with approximation factor less than $\frac{3}{2}$. This holds even if all job processing times are from $\{1, 2, \infty\}$; in contrast the optimum schedule can be computed in polynomial time if all processing times are from $\{1, 2\}$. Lenstra et al. [18] also present a polynomial-time approximation algorithm, based on linear programming, for the optimum schedule with approximation factor 2.

Kleinberg [13] formulated the problem of finding a restricted schedule with minimum social cost for jobs on unrelated parallel machines as a *single-source unsplittable flow* problem. Using this formulation, Kolliopoulos and Stein [15] provided a polynomial time 3-approximation algorithm for the general case, and they also presented an approximation algorithm with approximation factor $2 - \frac{1}{C}$ for the special case where all processing times are from $\{p, Cp, \infty\}$ for any $C > 1$ and $\frac{1}{C} \geq p > 0$. All results in [13, 15] are based on rounding techniques. For the general case, Dinitz et al. [3] give a 2-approximation algorithm which directly computes an unsplittable flow from a splittable flow and uses time $O(nm)$, assuming that a splittable flow solution is available. Our algorithm has an approximation factor of $2 - \frac{1}{w}$ if the traffics are from $\{1, 2, \ldots, w\}$. We use techniques as in the generic PREFLOW-PUSH algorithm [8, 10] for computing a maximum flow in a network. For an overview of algorithms to compute flows in networks, see Ahuja et al. [1] and the introduction from Goldberg et al. [9].

The *KP-model* [17] for routing selfish users on parallel links, and its Nash equilibria, were studied extensively in the last years; see, for example, [2, 4, 6, 7, 16, 19], and [5] for a recent survey.

For the case of related links, Graham's *LPT* scheduling algorithm [11] computes a pure Nash equilibrium in the KP-model [6]. However, computing the (pure) Nash equilibrium with minimum social cost is $\mathcal{NP}$-hard [6]. The Nashification technique is due to Gairing et al. [7]; polynomial-time Nashification algorithms for the cases of identical links and related links have been presented by Gairing et al. [7] and by Feldmann et al. [4], respectively. In order to compensate for the restrictions imposed by the sets of allowed links, our Nashification algorithm in Section 4 takes a completely different approach than those by employing techniques from flow algorithms.

The *coordination ratio,* also known as *price of anarchy* [22], was first introduced and studied by Koutsoupias and Papadimitriou [17]. For the case of arbitrary users and identical links, a *tight* bound of $\Theta\left(\frac{\log m}{\log\log m}\right)$ on coordination ratio has been shown by Czumaj and Vöcking [2] and by Koutsoupias et al. [16]; tightness follows from a lower bound observed in [17]. The coordination ratio reduces to $2 - \frac{2}{m+1}$ when restricted to pure Nash equilibria [7]. For the case of related links, Czumaj and Vöcking prove that the coordination ratio is $\Theta\left(\min\{\frac{\log m}{\log\log m}, \log\frac{c_1}{c_m}\}\right)$ for the case of pure Nash equilibria and $\Theta\left(\frac{\log m}{\log\log\log m}\right)$ for the case of arbitrary (mixed) Nash equilibria. Our bounds on coordination ratio (for pure Nash equilibria) imply that the impact of restricted schedules on coordination ratio is a multiplicative factor of either $\Theta\left(\frac{\log m}{\log\log m}\right)$ for the case of identical links, or $\Theta(m)$ for the case of related links.

Road Map: The rest of this paper is organized as follows. In Section 2, we introduce the model of selfish routing on restricted parallel links. Our approximation algorithms for a schedule with minimum social cost are presented in Section 3. Section 4 presents our Nashification algorithm for the case of identical links. Our bounds on coordination ratio appear in Section 5.

2. MODEL

We consider a *network* consisting of a set of m parallel *links* $1, 2, \ldots, m$ from a *source* node to a *destination* node. Each of n *network users* $1, 2, \ldots, n$, or *users* for short, wishes to route a particular amount of traffic along a (non-fixed) link from source to destination. For each user i denote the *strategy set* $A_i \subseteq [m]$ as the set of links to which user i can possibly be assigned and let $A = \sum_{i\in[n]} |A_i|$ be the total number of strategies. Denote w_i the *traffic* of user $i \in [n]$. Assume, without loss of generality, that $w_1 \geq \ldots \geq w_n$, and denote $W = \sum_{i=1}^{n} w_i$ the *total traffic.* A *pure strategy* for user $i \in [n]$ is some specific link. An *assignment* α is represented by an n-tuple $\langle l_1, l_2, \ldots, l_n\rangle \in [m]^n$ where $l_i \in A_i$ is the strategy of user i. In other papers on Nash equilibria, assignments are also called *pure strategy profiles.*

Denote $c_j > 0$ the *capacity* of link $j \in [m]$, representing the rate at which the link processes traffic. In the model of *identical capacities,* all link capacities are equal. Link capacities may vary arbitrarily in the model of *related capacities.* Without loss of generality assume $c_1 \geq \ldots \geq c_m$, and denote $C = \sum_{j=1}^{m} c_j$ the *total capacity.* So, the *latency* for traffic w_i through link j equals $\frac{w_i}{c_j}$. For an assignment $\langle l_1, l_2, \ldots, l_n \rangle$, the *load* on link j is defined by the sum of the traffics assigned to j, that is $\delta_j = \sum_{k:l_k=j} w_k$. The *latency for user* $i \in [n]$, denoted λ_i, is the latency of the link it chooses, that is, $\lambda_i = \frac{\delta_{l_i}}{c_{l_i}}$. Associated with a traffic vector $\mathbf{w}$ and an assignment α is the *social cost* [17, Section 2], denoted $\mathsf{SC}(\mathbf{w}, \alpha)$, which is the maximum (over all links) latency of traffic through a link; thus,

$$\mathsf{SC}(\mathbf{w}, \alpha) = \max_{j \in [m]} \frac{\sum_{k:l_k=j} w_k}{c_j}.$$

The *social optimum* [17, Section 2] associated with a traffic vector $\mathbf{w}$, denoted $\mathsf{OPT}(\mathbf{w})$, is the *least possible* maximum (over all links) latency of traffic through a link. Note that both $\mathsf{SC}(\mathbf{w}, \alpha)$ and $\mathsf{OPT}(\mathbf{w})$ also depend on the capacities and the strategy sets.

We are interested in a special class of assignments called Nash equilibria [21] that we describe below. A user i is *satisfied* if it has no incentive to unilaterally change its strategy, that is,

$$\lambda_{l_i} \leq \lambda_j + \frac{w_i}{c_j} \quad \text{for all} \quad j \in A_i.$$

If a user can *improve* by changing its strategy, we call it *unsatisfied.* An assignment α is a *Nash equilibrium* [17, Section 2] if all users $i \in [n]$ are satisfied. The *coordination ratio* [17] is the maximum value, over all traffic vectors $\mathbf{w}$ and Nash equilibria α of the ratio $\mathsf{SC}(\mathbf{w}, \alpha)/\mathsf{OPT}(\mathbf{w})$.

Throughout the paper we assume that all user traffics w_i and link capacities c_j are integer.

3. APPROXIMATION OF OPTIMUM SOCIAL COST

In this section we show, how we can compute an assignment (not necessarily an equilibrium one) that approximates an assignment with minimum social cost. We adapt the generic PREFLOW-PUSH algorithm for computing a maximum flow. Our first result considers the case that link capacities are identical. Here we show an approximation factor of $2 - \frac{1}{w_1}$. Afterwards we show how this result can be generalized to the related link model, with a slight increase of the approximation factor to 2.

3.1 Identical Links

We start with the model of identical links with restricted assignments. We introduce a residual network G_α representing a partial assignment α.

DEFINITION 3.1. *Given a partial assignment α, we define a directed bipartite graph $G_\alpha = (V, E_\alpha)$ where $V = L \cup U$ and each link is represented by a node in L whereas each user defines a node in U. Furthermore, $E_\alpha = E_\alpha^1 \cup E_\alpha^2$ with*

$$\begin{aligned} E_\alpha^1 &= \{(u,v) : u \in L, v \in U, u = l_v\} \text{ ,and} \\ E_\alpha^2 &= \{(u,v) : u \in U, v \in L, v \in A_u - \{l_u\}\}. \end{aligned}$$

We use an integer a to control the approximation of an optimal assignment. The intention is to find an a, which is a lower bound on $\mathsf{OPT}(\mathbf{w})$ and then to compute an assignment with $\mathsf{SC}(\mathbf{w}, \alpha) \leq a + w_1$. For any integer a, we partition the set of links L in three subsets:

$$\begin{aligned} L^- &= \{j : \delta_j(\alpha) \leq a\} \\ L^0 &= \{j : a + 1 \leq \delta_j(\alpha) \leq a + w_1\} \\ L^+ &= \{j : \delta_j(\alpha) \geq a + w_1 + 1\} \end{aligned}$$

In this setting we do not have a dedicated source or sink. However, at each time nodes in L^+ and L^- can be interpreted as source and sink nodes, respectively. Note, that those sets change over time. As the generic PREFLOW-PUSH algorithm we use a height function $h : V \to \mathbb{N}$ with the property that $h(u) = 0$, for all $u \in L^-$ and $h(u) \leq h(v) + 1$ for every edge $(u, v) \in E_\alpha$. For a given integer a define the excess flow $e(i)$ into node $i \in V$ as follows

$$e(i) = \begin{cases} 1 & \text{, if } i \in L^+ \\ 1 & \text{, if } i \in U \wedge \nexists j \in L : (j,i) \in E_\alpha \\ 0 & \text{, otherwise.} \end{cases}$$

We call a node $u \in V$ overflowing, if $e(u) = 1$.

As the generic PREFLOW-PUSH we use two basic operations PUSH and LIFT. A node $u \in V$ can apply PUSH(u, v), if u is overflowing, $(u, v) \in E_\alpha$ and $h(u) = h(v) + 1$. Furthermore, for a node $u \in L$ it must hold, that $h(u) \leq 2m$. The operation PUSH(u, v) then deletes (u, v) from and adds (v, u) to G_α. In other words we change the direction of (u, v), thus each PUSH is *saturating.* Afterwards $e(u)$ and $e(v)$ are recomputed. If $u \in L$ then PUSH(u, v) has the effect that user v gets unassigned from link u (afterwards v is not assigned to any link). On the other hand if $u \in U$ then PUSH(u, v) assigns an unassigned user u to link v.

```
PUSH$(u, v)$

Applies when:
$e(u) > 0$, $h(u) = h(v) + 1$, $(u, v) \in E_\alpha$
and if $u \in L$ then $h(u) \leq 2m$
{
    $E_\alpha \leftarrow E_\alpha - \{(u, v)\} + \{(v, u)\}$
    update $e(u)$, $e(v)$
}

LIFT$(u)$

Applies when:
$e(u) > 0$, $\forall (u, v) \in E_\alpha : h(u) \leq h(v)$
and if $u \in L$ then $h(u) \leq 2m$
{
    $h(u) \leftarrow 1 + \min\{h(v) : (u, v) \in E_\alpha\}$
}
```

Figure 1: Push and Lift

A node $u \in V$ can apply LIFT(u), if u is overflowing, and $\forall (u, v) \in E_\alpha : h(u) \leq h(v)$. Furthermore, for a node $u \in L$ it must hold, that $h(u) \leq 2m$. If this is the case, then LIFT(u) increases the height of u to $h(u) \leftarrow 1 + \min\{h(v) : (u, v) \in E_\alpha\}$.

The goal of algorithm UNSPLITTABLE-PREFLOW-PUSH in Figure 2 is to compute an assignment of users to links, such that for all links $j \in [m] : \delta_j \leq a + w_1$ and $a < \mathsf{OPT}(\mathbf{w})$. UNSPLITTABLE-PREFLOW-PUSH() first initializes the height function $h(u) = 0$ and computes the excess $e(u)$ for each

```
UNSPLITTABLE-PREFLOW-PUSH($G_\alpha, a$)
for each node $u \in V$
    $h(u) = 0$; compute $e(u)$
while there exists an applicable
        PUSH or LIFT operation
        and there exists $u \in L^-$
    do select applicable PUSH or
        LIFT operation and perform it
```

Figure 2: Unsplittable-Preflow-Push

node $u \in V$. Then it performs PUSH and LIFT operations as long as there are such applicable operations and as long as there are still nodes in L^-. From the conditions under which PUSH and LIFT are applicable, we derive the following observation:

OBSERVATION 3.1. *When* UNSPLITTABLE-PREFLOW-PUSH *terminates then either $e(u) = 0$ for all $u \in V$, or $h(u) > 2m$ for all $u \in L$ with $e(u) = 1$, or $L^- = \emptyset$.*

LEMMA 3.1. UNSPLITTABLE-PREFLOW-PUSH *performs at most $O(m(n+m))$* LIFT *and $O(mA)$* PUSH *operations.*

PROOF. LIFT always increases the height $h(u)$ of some node u. Consider a link $u \in L$. Link u can only LIFT if $h(u) \leq 2m$. At the beginning $h(u) = 0$, which implies that u can LIFT at most $2m+1$ times. Now consider a user $u \in U$. We show, that if LIFT(u) leaves u with $h(u) > 2m+1$, then u will never LIFT again. Since we consider a bipartite graph u has only edges (u,v) to links $v \in L$. If LIFT(u) increases the height of u to some value $h(u) > 2m+1$ then $\min\{h(v) : (u,v) \in E_\alpha\} > 2m$. But then v can never again apply a LIFT(v). It follows, that u can lift at most $2m+2$ times. We have $m+n$ nodes and each node performs at most $2m+2$ LIFT operations. This proves the bound on the number of LIFT operations.

For any edge $(u,v) \in E_\alpha$ consider the number of PUSH operations from u to v and from v to u. By the time that PUSH(u,v) was executed $h(u) = h(v)+1$. Also (u,v) was deleted from E_α and (v,u) was added. Before PUSH(v,u) can be executed $h(v)$ must increase by at least 2. Therefore v first has to LIFT. It follows, that in between any two PUSH operations between u and v there must be a LIFT of u or v. This shows, that the number of PUSH operations per edge is bounded by $O(m)$. Since A is the number of edges in G_α UNSPLITTABLE-PREFLOW-PUSH performs at most $O(mA)$ PUSH operations. $\square$

LEMMA 3.2. *When* UNSPLITTABLE-PREFLOW-PUSH *terminates and $\exists u \in V : e(u) > 0$, then $\mathsf{OPT}(\mathbf{w}) > a+1$.*

PROOF. There are two cases to consider here. Either UNSPLITTABLE-PREFLOW-PUSH terminated since $L^- = \emptyset$ or $L^- \neq \emptyset$ and $\forall u \in V$ with $e(u) > 0$ we have $h(u) > 2m$. First assume $L^- = \emptyset$. We know that $\exists u \in V$ with $e(u) > 0$ and therefore $\exists u \in L : \delta_u > a + w_1$ which immediately implies that $\mathsf{OPT}(\mathbf{w}) > a+1$. Now let $L^- \neq \emptyset$. Since UNSPLITTABLE-PREFLOW-PUSH terminated we know that $h(v) > 2m$ for all v with $e(v) > 0$. Since G_α is bipartite, the maximum length of a path between two nodes in L is at most $2m$. If there is a path from v to some $u \in L^-$ then for each edge (i,j) on this path $h(i) \leq h(j)+1$ and thus $h(v) \leq 2m$. This implies that there is no path from v to any node $u \in L^-$. Let $B \subset L$ be the set links that are still reachable from node v. No user that is assigned to a link in B can be assigned to a link outside B (otherwise there would be a path to this link). But $\delta_i \geq a+1, \forall i \in B$ and $\delta_v > a + w_1$. It follows, that $\mathsf{OPT}(\mathbf{w}) > a+1$. $\square$

We now show how to use UNSPLITTABLE-PREFLOW-PUSH to approximate a routing with minimum social cost. We do this by a series of calls to UNSPLITTABLE-PREFLOW-PUSH(G_α, a). With a binary search on $a \in [0, W], a \in \mathbb{N}$, we find the smallest a, such that UNSPLITTABLE-PREFLOW-PUSH(G_α, a) returns an assignment with $\delta_j \leq a + w_1, \forall j \in [m]$, that is $e(u) = 0, \forall u \in V$. Since a is the smallest value for which this holds, UNSPLITTABLE-PREFLOW-PUSH$(G_\alpha, a-1)$ returns with $e(u) > 0$ for some $u \in V$. It follows from Lemma 3.2 that $\mathsf{OPT}(\mathbf{w}) > a$. Since also $\mathsf{OPT}(\mathbf{w}) \geq w_1$, this implies, that

$$\frac{\mathsf{SC}(\mathbf{w}, \alpha)}{\mathsf{OPT}(\mathbf{w})} \leq \frac{a + w_1}{\mathsf{OPT}(\mathbf{w})} \leq \frac{\mathsf{OPT}(\mathbf{w}) - 1 + w_1}{\mathsf{OPT}(\mathbf{w})} \leq 2 - \frac{1}{w_1}.$$

The following theorem follows immediately.

THEOREM 3.3. *There exists an algorithm, that approximates a routing on identical links with minimum social cost within a factor of $2 - \frac{1}{w_1}$ in time $O(mA\log(W))$.*

PROOF. Lemma 3.1 shows that at most $O(mA)$ PUSH and $O(m(n+m)$ LIFT operations are possible. Using standard techniques [1] from PREFLOW-PUSH, this can be done in time $O(mA)$. The binary search on $a \in [0, W]$ contributes a factor of $\log(W)$. $\square$

3.2 Related Links

We now turn our attention to the related link model with restricted assignments. We show how the approximation algorithm from the previous section can be generalized to this model.

Again for an assignment α we define G_α similar to Definition 3.1. The set of nodes is the same as in Definition 3.1, but we consider a subset of the edges, namely all edges $\{u, v\}$ with $u \in U$ and $v \in L$, such that $w_u \leq a \cdot c_v$. We will see later, why we can make this restriction on G_α. We adapt the partition of the links according to a positive rational number a as follows:

$$\begin{aligned} L^- &= \{j : \delta_j(\alpha) \leq ac_j\} \\ L^0 &= \{j : ac_j < \delta_j(\alpha) \leq 2ac_j\} \\ L^+ &= \{j : \delta_j(\alpha) > 2ac_j\} \end{aligned}$$

The excess $e(u)$ is defined as in Section 3.1, however the partition of L and the graph G_α changed. We use exactly the same algorithm as in Figure 1 and Figure 2. The adaption does not influence the proof of Lemma 3.1. We proceed by proving a lemma, comparable to Lemma 3.2.

LEMMA 3.4. *When* UNSPLITTABLE-PREFLOW-PUSH *terminates and $\exists u \in V : e(u) > 0$, then $\mathsf{OPT}(\mathbf{w}) > a$.*

PROOF. We have to consider the same two cases as in Lemma 3.2. If $L^- = \emptyset$ then the lemma follows immediately from the Definition of L^-. So assume, that $L^- \neq \emptyset$ and $\forall v \in V$ with $e(v) > 0$ we have $h(v) > 2m$. This again implies that there is no path from v to some node $u \in L^-$ in G_α (see Lemma 3.2). Let $B \subset L$ be the set of links that are still reachable from node v. Since for G_α we considered only edges $\{u,v\}$ with $u \in U, v \in V$ and $w_u \leq a \cdot c_v$, it is now possible that a user that is assigned to a link in B could be assigned to a link $p \notin B$. However on link p it would cause latency $\lambda_p > a$. On the other hand, if we do not move users from links in B to links outside B, then $\lambda_u > a$ for all $u \in B$. In either case $\mathsf{OPT}(\mathbf{w}) > a$ □

Again we make a series of calls to Unsplittable-Preflow-Push(G_α, a) and we compute the smallest a, such that $L^+ = \emptyset$ in the computed assignment. However, this time we do the binary search for a on the set of points $\{\frac{1}{c_1}, \ldots, \frac{W}{c_1}\} \cup \ldots \cup \{\frac{1}{c_m}, \ldots, \frac{W}{c_m}\}$. These are at most mW points and they include all possible values of link latencies. After the binary search, Unsplittable-Preflow-Push(G_α, a) returned an assignment with $L^+ = \emptyset$ and therefore $\mathsf{SC}(\mathbf{w}, \alpha) \leq 2a$. On the other hand let a' be the largest value with $a' < a$ in our sample space. Then Unsplittable-Preflow-Push(G_α, a') returned an assignment with $L^+ \neq \emptyset$. By Lemma 3.4 this directly implies that $\mathsf{OPT}(\mathbf{w}) > a'$ and thus $\mathsf{OPT}(\mathbf{w}) \geq a$. The following theorem follows immediately.

Theorem 3.5. *There exists an algorithm that approximates a routing on related links with minimum social cost within a factor of 2 in time $O(mA \log(mW))$.*

Proof. One call to Unsplittable-Preflow-Push() can be done in time $O(mA)$ (see Theorem 3.3). The binary search on $a \in \{\frac{j}{c_i} : 1 \leq j \leq W, 1 \leq i \leq m\}$ can be implemented to run in $O(\log(mW))$ time. □

Other approaches that solve this problem use methods from linear programming [18] or they compute an unsplittable flow using rounding techniques [13, 14] or directly compute an unsplittable flow from a splittable flow [3]. Our combinatorial algorithm is simpler and improves on the approximation quality if we restrict to identical links.

4. NASHIFICATION

In Section 3.1 we gave an algorithm that approximates an optimal routing up to a constant factor on *identical links*. However, this algorithm does not necessarily compute a Nash equilibrium. The approximation algorithm returns an assignment with $\delta_j \leq a + w_1, \forall j \in [m]$ and $\mathsf{OPT}(\mathbf{w}) > a + 1$, which does not imply that w_1 is satisfied. However, w_1 would be satisfied, if $a < \delta_j \leq a + w_1$ holds for all $j \in [m]$. We have already seen in Section 3.1 that this is not always possible, and we will show in the following, how to find an assignment satisfying w_1. We then fix the assignment of all users with traffic w_1 and proceed with the next smaller traffic. This is done until all users are satisfied. In each step we make sure, that fixed users stay satisfied. The algorithm, called Unsplittable-Blocking-Flow(), combines ideas from blocking flows with the idea of pushing users without splitting them. We adapt the definitions of L^-, L^0 and L^+ from Section 3.1 by replacing w_1 by an arbitrary traffic size w.

For a total assignment α, that is each user is assigned to a link, we use the graph G_α from Definition 3.1 to define a graph $G_\alpha(w)$ where V stays the same, but from E_α we now only consider edges $E_\alpha(w) = E_\alpha - \{(u,v) : u \in U, v \in V, w_u > w\}$. This means that users u with $w_u > w$ stay assigned to their link. We will use G_α instead of $G_\alpha(w)$ if it is clear from the context, which w is used.

We will now present Algorithm Unsplittable-Blocking-Flow() that will shift users so that the latencies of links from L^- are never decreased, the latencies of links from L^+ are never increased, and links from L^0 stay in L^0. Our algorithm is controlled by a height function $h : V \to \mathbb{N}_0$ with $h(j) = \mathrm{dist}_{G_\alpha}(j, L^-), \forall j \in V$. We call an edge (u,v) admissible, if $h(u) = h(v) + 1$. In an admissible path all edges are admissible. For each node $j \in V$ with $0 < h(j) < \infty$ let $S(j)$ be the set of successors of node j, that is the set of nodes to which j has an admissible edge, i.e.

$$S(j) = \{i \in V : (j,i) \in E_\alpha \text{ and } h(j) = h(i) + 1\}.$$

Note that $S(j)$ also defines the set of admissible edges leaving j. Let $s(j)$ be the first node on list $S(j)$.

Definition 4.1. *A link $j \in L$ with $0 < h(j) < \infty$ is called helpful if $\delta_j(\alpha) \geq a + 1 + w_{s(j)}$.*

Lemma 4.1. *Let v_0 be a helpful link of minimum height. Then there exists a sequence $v_0, \ldots, v_r$ where $v_{2i} \in L, \forall 0 \leq i \leq r/2$ and $v_{2i+1} \in U, \forall 0 \leq i < r/2$ with the following properties:*

(a) $(v_i, v_{i+1}) \in E_\alpha$ *and* $h(v_i) = h(v_{i+1}) + 1$

(b) $\delta_{v_0} \geq a + 1 + w_{s(v_0)}$

(c) $a + 1 \leq \delta_{v_{2i}} + w_{s(v_{2i-2})} - w_{s(v_{2i})} \leq a + w, \forall 0 < i < r/2$

(d) $\delta_{v_r} + w_{s(v_{r-2})} \leq a + w$

Proof. By the definition of a helpful link $0 < h(v_0) < \infty$ and thus there exists a path from v_0 to a link in L^- that defines the height of v_0. On this path (a) must hold. Furthermore, condition (b) follows directly from the definition of a helpful link.

Note that a link $j \in L^+$ is helpful if $h(j) < \infty$. So if we start a path with a helpful link of minimum height, then all link nodes $v_2, v_4, \ldots, v_{r-2}$ belong to L^0 and v_r may belong to L^0 or L^-. Therefore

$$\delta_{v_{2i}} \leq a + w, \forall 0 < i \leq r/2.$$

Furthermore, none of these nodes is helpful, which implies that

$$\delta_{v_{2i}} < a + 1 + w_{s(v_{2i})}, \forall 0 < i \leq r/2.$$

There are two cases to consider now. If $\delta_{v_{2i}} + w_{s(v_{2i-2})} \leq a + w$ then $v_r = v_{2i}$ and condition (d) holds. On the other hand, since $w \geq w_{s(v_{2i})}$

$$\begin{aligned} \delta_{v_{2i}} + w_{s(v_{2i-2})} &\geq a + w + 1 \\ \Rightarrow \quad \delta_{v_{2i}} + w_{s(v_{2i-2})} - w_{s(v_{2i})} &\geq a + w + 1 - w_{s(v_{2i})} \\ \Rightarrow \quad \delta_{v_{2i}} + w_{s(v_{2i-2})} - w_{s(v_{2i})} &\geq a + 1, \end{aligned}$$

proving the lower bound in (c). To proof the upper bound, note that v_{2i} is not helpful $\forall 0 < i \leq r/2$. It follows that

$$\begin{aligned} \delta_{v_{2i}} &< a + 1 + w_{s(v_{2i})} \\ \Rightarrow \quad \delta_{v_{2i}} - w_{s(v_{2i})} + w_{s(v_{2i-2})} &\leq a + w_{s(v_{2i-2})} \\ \Rightarrow \quad \delta_{v_{2i}} + w_{s(v_{2i-2})} - w_{s(v_{2i})} &\leq a + w, \end{aligned}$$

proving the upper bound in (c). This completes the proof of the lemma. □

We are now ready to present Algorithm Unsplittable-Blocking-Flow(). Algorithm Unsplittable-Blocking-Flow() stops, when $L^- = \emptyset$ or $\forall v \in L^+ : h(v) = \infty$. The

```
UNSPLITTABLE-BLOCKING-FLOW($\alpha, a, w$)
Input:  assignment $\alpha$
        positive integers $a, w$
Output:  assignment $\beta$

compute $h$
while $L^- \neq \emptyset$ and $\exists v \in L^+ : h(v) < \infty$
{
  $d = \min_{v \in L^+}(h(v))$
  while $\exists$ admissible path from
        $v \in L^+, h(v) = d$ to $L^-$
  {
    choose helpful link $v$ of
        minimum height
    push users along helpful path
        defined by $v$
    adjust $\alpha, G_\alpha$
  }
  recompute $h$
}
return $\alpha$
```

Figure 3: Unsplittable-Blocking-Flow

algorithm works in phases. Before the first phase starts, the height function h is computed as the distance in G_α of each node to a node in L^-. In each phase first the minimum height $d = h(v)$ of a node $v \in L^+$ is computed. Inside a phase we do not update the height function, but we successively choose a helpful link v of minimum height and we push users along the helpful path induced by v and adjust the assignment accordingly. In order to update G_α we have to change the direction of two arcs for each user push. The phase ends, when no further admissible path from an node $v \in L^+$ with $h(v) = d$ to some node in L^- exists. Before the new phase start we recompute h and we check whether we have to start a new phase or not.

LEMMA 4.2. *Let β be the assignment computed by* UNSPLITTABLE-BLOCKING-FLOW*(α, a, w) then*

(a) $j \in L^-(\alpha) \Rightarrow \delta_j(\beta) \geq \delta_j(\alpha)$
(b) $j \in L^0(\alpha) \Rightarrow a + 1 \leq \delta_j(\beta) \leq a + w$
(c) $j \in L^+(\alpha) \Rightarrow \delta_j(\beta) \leq \delta_j(\alpha)$.

PROOF. UNSPLITTABLE-BLOCKING-FLOW() only pushes users along a helpful path that is defined by a helpful link v of minimum height $h(v)$. Lemma 4.1 shows that by doing this we never add a link to L^-. Furthermore, a link in L^- can only be the last link in such a helpful path. But this link only receives load which implies (a). On the other hand a link $v \in L^+$ can only be the first link in such a helpful path, (c) follows. Condition (b) also follows directly from Lemma 4.1. □

As an immediate consequence of Lemma 4.2 we get the following corollary:

COROLLARY 4.3. *Let β be the assignment computed by* UNSPLITTABLE-BLOCKING-FLOW*(α, a, w) then*

$$\max_{j \in [m]} \delta_j(\beta) \leq \max_{j \in [m]} \delta_j(\alpha) \text{ ,and}$$
$$\min_{j \in [m]} \delta_j(\beta) \geq \min_{j \in [m]} \delta_j(\alpha).$$

LEMMA 4.4. *Let β be the assignment computed by* UNSPLITTABLE-BLOCKING-FLOW*(α, a, w) then one of the following 3 conditions is true*

(a) $L^-(\beta) = \emptyset$
(b) $L^+(\beta) = \emptyset$
(c) $\exists B \subset [m]$ *such that*

  (c1) $\delta_j(\beta) \geq a + 1$ *,* $\forall j \in B$*, and*
  (c2) $\delta_j(\beta) \leq a + w$ *,* $\forall j \in [m] - B$*, and*
  (c3) $l_i \in B \Rightarrow A_i \subseteq B, \forall i \in [n]$ *with* $w_i \leq w$.

PROOF. If either (a) or (b) holds then the algorithm terminates. So assume that $L^-(\beta) \neq \emptyset$ and $L^+(\beta) \neq \emptyset$ and the algorithm terminates. It follows that $\forall v \in L^+ : h(v) = \infty$ which implies that in G_β there is no path from a link in L^+ to a link in L^-. Define B to be the set of links that are reachable from some link in L^+. Since L^- is not reachable from a link in L^+, (c1) holds. All links in L^+ are also in B, therefore (c2). By the definition of B, if a user i is assigned to a link $l_i \in B$, then it can not be assigned to a link in $[m] - B$ which implies (c3). This completes the proof of the lemma. □

THEOREM 4.5. UNSPLITTABLE-BLOCKING-FLOW*() can be implemented to run in $O(m \cdot A)$ time.*

PROOF. We consider a phase of algorithm UNSPLITTABLE-BLOCKING-FLOW() to be a single pass through the outer while-loop. In a phase we compute the minimum height $d = \min_{v \in L^+}(h(v))$ of a link in L^+. Then we successively choose a helpful link v of minimum height and we push users along the helpful path induced by v by changing the direction of each arc along this path. This can make other nodes helpful. We will see later, how the problem of finding the next helpful link of minimum height is solved. We push users along such helpful paths until no further helpful path from a link $v \in L^+$ with $h(v) = d$ exists. We only change the direction of admissible edges, therefore if no further such path exists, then we reached a blocking flow. Thus, after updating the height function, the minimum height d of a link in L^+ increased (see Ahuja et al. [1]). Since we have a bipartite graph, d has to increase by at least 2 and furthermore the maximum height of any link v with admissible path to a link in L^- is at most $2m$. This implies, that the number of phases is a most m.

Note, that A is an upper bound on the number of edges in $G_\alpha(w)$. Therefore computing the height function h can be done in $O(A)$ time using *breadth-first search.* While computing the height function we built a graph G_{ad} that only contains nodes and edges that are on an admissible path from any node u with $h(u) = d$ to any node v with $h(v) = 0$. Furthermore, every node v saves the set of nodes defining its height in list $S(v)$ and every node keeps track of its positions in this lists. Note that each entry in a list defines an admissible edge. Even if $S(u)$ holds all possible neighbors for a path to a link in L^-, we always choose the first entry $s(u)$ from $S(u)$ to define a helpful path. This is done to bound the running time. Inside a phase we always push users along a path that is induced by a helpful link v of minimum height $h(v)$. There are two ways, how a link v' with $h(v') \leq h(v)$ can become helpful. Either the load $\delta_{v'}$ or the first successor $s(v') \in S(v')$ changed. Consider a link v' on the path induced by v. Link v' can become helpful, since both the load $\delta_{v'}$ and $s(v')$ have changed. Furthermore, it is possible that $S(v')$ became empty. In this case v' is no longer on an admissible path to a link in L^- and therefore v' and all edges (u, v') entering v' are deleted from G_{ad}. So, v' is extracted from $S(u)$. This again can make u helpful, since $s(u)$ may have changed. It is also possible that $S(u)$ became empty and we have to proceed recursively.

We now show, how to find the next helpful link of minimum height efficiently using a stack K. Intuitively, K holds all nodes that could be helpful or could make other nodes helpful. To simplify the notation we say that user u is helpful, if u is not assigned to any link. Since we already used Push in our algorithm, we will use Write and Read for the basic stack operations. At the beginning of each phase every link could be helpful. We initialized K with all nodes from $u \in G_{ad}$, ordered by the height $h(u)$, such that on the top of K there is a node u with $h(u) = 0$. A *level* on the stack is a set of successive entries where the height stays the same. We now do the following until K becomes empty:

- We first take node u =Read from K.
- If $s(u)$ is not defined, then we define $s(u)$ as the next element from $S(u)$.
- If $s(u)$ is now defined and if u is a helpful link then we Push$(u, s(u))$, we delete the edge $(u, s(u))$ from G_{ad}, we Write(u) to K (since u may still be helpful) and we Write$(s(u))$ to K (since $s(u)$ may now be helpful).
- Else if $s(u)$ is not defined, then u is no longer on an admissible path to a link in L^-. In this case $\forall (v, u) \in G_{ad}$ we delete (v, u) from G_{ad} (since (u, v) is no longer admissible) and we Insert(v) to the previous level (v may be helpful now). Here Insert(v) means that v is inserted into K such that the order of the heights on K is preserved. This can be done in constant time.

If K becomes empty, then the phase end. At the beginning there are $n + m$ nodes on K. Each time that we Read from K, we also delete at least one edge from G_{ad}. And each time that we delete one edge from G_{ad}, we add at most two nodes to K. Since A is an upper bound on the number of edges in G_{ad}, we know that the time that the algorithm spends per phase is bounded by $O(n + m + A) = O(A)$. $\square$

We will now show how algorithm Unsplittable-Blocking-Flow() can be used to convert a given assignment α into a pure Nash equilibrium β with non-increased social cost. Let $\widetilde{w}_1 > \ldots > \widetilde{w}_r$ be all different user traffics from $w_1, \ldots, w_n$. The idea is to compute a sequence of assignments $\beta_0, \ldots, \beta_r$ such that $\beta_0 = \alpha$, and in $\beta_i, 1 \leq i \leq r$, all users j with $w_j \geq \widetilde{w}_i$ are satisfied. We call the computation of β_i from β_{i-1} *stage i*. The aim in stage i is to compute an assignment β_i from β_{i-1}, such that in β_i all users u with $w_u \geq \widetilde{w}_i$ are satisfied. In stage i we set $w = \widetilde{w}_i$.

We first define stage 1. By a series of calls to Unsplittable-Blocking-Flow() we compute an assignment where L^- and L^+ are either both empty or both not empty. We do this by binary search on $a \in [\min_j \delta_j(\beta_0), \max_j \delta_j(\beta_0)], a \in \mathbb{N}$ as follows: If Unsplittable-Blocking-Flow() returns an assignment with $L^- = \emptyset$ and $L^+ \neq \emptyset$, then we increase a. On the other hand, if Unsplittable-Blocking-Flow() returns an assignment with $L^- \neq \emptyset$ and $L^+ = \emptyset$, then we decrease a. If after the binary search $L^- = \emptyset$ and $L^+ = \emptyset$, then we have computed an assignment where all users with traffic at least $\widetilde{w}_1$ are satisfied. If neither $L^- = \emptyset$ nor $L^+ = \emptyset$ it follows that condition (c) from Lemma 4.4 holds. Define B as in Lemma 4.4. In this case we split our instance into two parts. One part with all links in B and all users that are currently assigned to a link in B, the other part holds the complement. Condition (c) from Lemma 4.4 implies that no user u with $w_u \leq \widetilde{w}_1$ that is assigned to a link in B has a link from $\overline{B}$ in its set of possible strategies. Furthermore, by conditions (c1) and (c2) from Lemma 4.4, no user u with $w_u = \widetilde{w}_1$ that is assigned to a link in $\overline{B}$ can improve by moving to a link in B, and Corollary 4.3 shows that this property is preserved. We now recursively proceed with the binary search on a in both parts. For the part that corresponds to B we increase a, and in the other part we decrease a. At the end of stage 1 all parts $B_1, \ldots, B_{p(1)}$ are put together to form β_1. This completes the definition of stage 1. After stage 1 we can define bounds on the load of each link as follows. For each $B_k, k \in [p(1)]$, define a lower bound $\mathsf{Low}(B_k)$ on the load of all links from B_k as the last value for a after the binary search on a in B_k. Moreover, define an upper bound $\mathsf{Up}(B_k) = \mathsf{Low}(B_k) + \widetilde{w}_1$. Assume, that $B_1, \ldots, B_{p(1)}$ are ordered, such that $\mathsf{Low}(B_1) \geq \ldots \geq \mathsf{Low}(B_{p(1)})$. By Lemma 4.4 there are no edges from a link in B_k to a link in B_ℓ when $k < \ell$. Furthermore, after stage 1 all users with traffic $\widetilde{w}_1$ are satisfied, and they stay satisfied if the load on each link resides within the defined bounds.

We now describe stage $i > 1$. At the beginning of stage i we have an assignment β_{i-1}, where the links are partitioned into sets $B_1, \ldots, B_{p(i-1)}$ with $\mathsf{Up}(B_k) = \mathsf{Low}(B_k) + \widetilde{w}_{i-1}$, for all $k \in [p(i-1)]$, and no edges from a link in B_k to a link in B_ℓ exist for $k < \ell$. As an invariant during stage i, we always have an assignment with properties as shown in Figure 4: We have a partition of the links into sets $B_1, \ldots, B_p$ with $\mathsf{Low}(B_1) \geq \ldots \geq \mathsf{Low}(B_p)$ and $\mathsf{Up}(B_1) > \ldots > \mathsf{Up}(B_p)$. Furthermore, there are no edges from a link in B_k to a link in B_ℓ when $k < \ell$. Note that the number of partitions p changes over time. In stage i the lower bound on the load of a link only increases and the upper bound only decreases. This implies, that users with traffic at least $\widetilde{w}_{i-1}$ stay satisfied during stage i. Some sets of links $B_k, k < x$ have not been considered yet and fulfill $\mathsf{Up}(B_k) - \mathsf{Low}(B_k) = \widetilde{w}_{i-1}$. Moreover, some sets of links $B_k, k > y$ have been *processed* already and fulfill $\mathsf{Up}(B_k) - \mathsf{Low}(B_k) = \widetilde{w}_i$. Finally, we have sets $B_x, \ldots, B_y$ of *active* links, with $\widetilde{w}_i < \mathsf{Up}(B_k) - \mathsf{Low}(B_k) \leq \widetilde{w}_{i-1}$ for all $k \in [x, y]$ and $\mathsf{Low}(B_x) = \ldots = \mathsf{Low}(B_y)$. Note, that at the beginning of stage i, the links from $B_{p(i-1)}$ are active, and the remaining links have not been considered. By a *sweep* over the sets of active links we compute a new assignment with the same structure. However, either the number of processed links increases or new links become active during a sweep.

A sweep works as follows: The aim of a sweep is either to completely process all links in B_y by increasing the lower bound of all active links to $\mathsf{Up}(B_y) - \widetilde{w}_i$, or to make all links in B_{x-1} active by increasing the lower bound of all active links to $\mathsf{Low}(B_{x-1})$. In order to preserve the structure of our assignment, we choose $a = \min\{\mathsf{Up}(B_y) - \widetilde{w}_i, \mathsf{Low}(B_{x-1})\}$. First assume $a = \mathsf{Up}(B_y) - \widetilde{w}_i$. We apply Unsplittable-Blocking-Flow() to the sub-instance defined by the set B_x. Unsplittable-Blocking-Flow() returns an assignment where one of the following cases holds.
$\underline{L^+ = \emptyset:}$ Since all links in B_x have load at most $a + \widetilde{w}_i < \mathsf{Up}(B_{x+1})$ and Corollary 4.3 implies that this property is

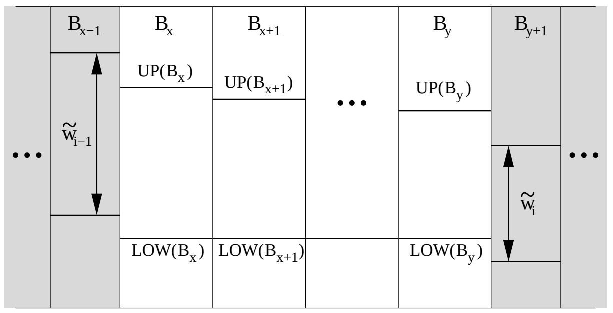

Figure 4: Sets of active links in stage i

preserved, we set $\mathsf{Up}(B_x) = \mathsf{Up}(B_{x+1})$ and merge B_x and B_{x+1}. Thus $B_{x+1} = B_x \cup B_{x+1}$.
$L^- = \emptyset$ and $L^+ \neq \emptyset$: In this case all links in B_x have load at least a and Corollary 4.3 implies that this property is preserved. Thus, we set $\mathsf{Low}(B_x) = a$.
$L^- \neq \emptyset$ and $L^+ \neq \emptyset$: Here, we split B_x according to condition (c) from Lemma 4.4 into sets B'_x and $\overline{B'}_x$. Condition (c3) implies, that there are no edges from a link in B'_x to a link in $\overline{B'}_x$. For B'_x we set $\mathsf{Low}(B'_x) = a$. For $\overline{B'}_x$ we set $\mathsf{Up}(\overline{B'}_x) = \mathsf{Up}(B_{x+1})$ and we merge $\overline{B'}_x$ with B_{x+1}. Thus $B_{x+1} = \overline{B'}_x \cup B_{x+1}$.

We iteratively proceed with $B_{x+1}, \ldots, B_y$. When we have reached B_y, $\mathsf{Low}(B_k) + \widetilde{w}_i \geq \mathsf{Up}(B_y)$ holds for all $k < y$. Thus, no user with traffic $\widetilde{w}_i$ can improve by moving from a link in B_y to a link in B_k, $k < y$. By Corollary 4.3, this property is preserved. Moreover, by Lemma 4.4 there are no edges from a link in B_y to a link in B_k, $k > y$. Thus, in order to satisfy users with traffic $\widetilde{w}_i$ (that were assigned to links in B_y), we make a binary search on $[\mathsf{Low}(B_y), a]$ in B_y as we did in the first stage for the whole instance. For the partitions $B_{y1}, \ldots B_{yp}$, that we receive we have $\mathsf{Up}(B_{yk}) = \mathsf{Low}(B_{yk}) + \widetilde{w}_i$, for all $k \in [p]$. Thus, these links are processed and become inactive.

Now, assume $a = \mathsf{Low}(B_{x-1})$. In this case all links in $B_{x-1}, \ldots, B_y$ are active. We do the same procedure as above. However, this time we start with B_{x-1}. Moreover, we have to treat B_y differently. We apply UNSPLITTABLE-BLOCKING-FLOW() to B_y with $a = \mathsf{Low}(B_{x-1})$. If $L^- = \emptyset$, then we set $\mathsf{Low}(B_y) = a$; if $L^+ = \emptyset$ then we set $\mathsf{Up}(B_y) = a + \widetilde{w}_i$ and do a binary search on $[\mathsf{Low}(B_y), a]$ as before; if $L^- \neq \emptyset$ and $L^+ \neq \emptyset$ then we split B_y according to condition (c) from Lemma 4.4 into sets B'_y and $\overline{B'}_y$. For B'_y we set $\mathsf{Low}(B'_y) = a$ and for $\overline{B'}_y$ we set $\mathsf{Up}(\overline{B'}_y) = a + \widetilde{w}_i$. Furthermore, we again do a binary search in $\overline{B'}_y$ as above, thus the links from $\overline{B'}_y$ are processed and become inactive. After each sweep, by renumbering the partitions, we get a new assignment that again has the same structure as in Figure 4. This completes the description of a sweep.

In each sweep we either add a set of links to the active links or we completely process some links. Stage i ends, when all links are processed. After stage i we have an assignment β_i, where the links are partitioned into sets $B_1, \ldots, B_{p(i)}$ with $\mathsf{Up}(B_k) = \mathsf{Low}(B_k) + \widetilde{w}_i$, for all $k \in [p(i)]$ and there are no edges from a link in B_k to a link in B_ℓ for $k < \ell$. Thus, all users with traffic $\widetilde{w}_i$ are satisfied. Since we only increased the lower bounds and decreased the upper bounds on the load of the links, all users u with traffic $w_u > \widetilde{w}_i$ stay satisfied during stage i. Hence we obtain:

LEMMA 4.6. *After stage i, every user u with traffic $w_u \geq \widetilde{w}_i$ is satisfied and stays satisfied until the end of stage r.*

THEOREM 4.7. *Consider the model of restricted strategies, arbitrary users and identical links. Then, it is possible to compute a Nash equilibrium in time $O(rmA(\log W + m^2))$, where r is the number of distinct traffic sizes.*

PROOF. Lemma 4.6 implies that after stage r *all* users are satisfied and therefore the resulting assignment β_r is a Nash equilibrium. It remains to show the running time of $O(rmA(\log W + m^2))$. One call to UNSPLITTABLE-BLOCKING-FLOW() with an instance of n users and m links can be done in $O(mA)$ time. Since in each sweep either a set of links becomes active or some links are completely processed, we have at most $O(m)$ sweeps per stage. Not counting the (possible) binary search at the end of a sweep, we do at most $O(m)$ calls to UNSPLITTABLE-BLOCKING-FLOW() per sweep. Since in a stage the binary search is done on distinct subsets of the users and links, the total time for the binary searches in a stage is $O(mA \log W)$. There are r stages, thus we get the running time of $O(rmA(\log W + m^2))$. □

5. COORDINATION RATIO

In this section we prove upper bounds on the coordination ratio for pure Nash equilibria. We first give an assignment in Example 5.1 yielding a lower bound (Lemma 5.1). We then use similar techniques as in [2] to prove two upper bounds for arbitrary users on identical links (Theorem 5.2) and identical users on related links (Theorem 5.3), which are asymptotically tight due to Lemma 5.1. We close by giving a tight upper bound for arbitrary users on related links.

EXAMPLE 5.1. *We consider the following instance of n identical users on m identical links:*

- *Let $\mathcal{M}_0, \ldots, \mathcal{M}_k$ be disjoint subsets of the m links with $|\mathcal{M}_0| = 1$ and $|\mathcal{M}_i| = (k-1)\prod_{1\leq j \leq i-1}(k-j)$ for all $2 \leq i \leq k$.*
- *Let $\mathcal{U}_0, \ldots, \mathcal{U}_{k-1}$ be disjoint subsets of the n users. $\mathcal{U}_0$ contains k users with strategy set $\mathcal{M}_0 \cup \mathcal{M}_1$, and $\mathcal{U}_i$ contains $(k-i)\cdot|\mathcal{M}_i|$ users with strategy set $\mathcal{M}_i \cup \mathcal{M}_{i+1}$ for $1 \leq i \leq k-1$.*

The assignment α is defined as follows: to link $\mathcal{M}_0$ we assign all k users in $\mathcal{U}_0$; to each link in $\mathcal{M}_i$ we assign $k-i$ users in $\mathcal{U}_i$ for all $2 \leq i \leq k-1$. The links in $\mathcal{M}_k$ remain empty.

LEMMA 5.1. *The assignment α given in Example 5.1 is a pure Nash equilibrium with*

$$\frac{\mathsf{SC}(\mathbf{w}, \alpha)}{\mathsf{OPT}(\mathbf{w})} > \Gamma^{-1}(m) - 2.$$

THEOREM 5.2. *Consider the model of arbitrary users with restricted strategy sets on identical links. Then, for any pure Nash equilibrium α, we have the tight bound*

$$\frac{\mathsf{SC}(\mathbf{w}, \alpha)}{\mathsf{OPT}(\mathbf{w})} = O\left(\frac{\log m}{\log \log m}\right).$$

PROOF. Consider a pure Nash equilibrium α with

$$k \cdot \mathsf{OPT}(\mathbf{w}) \leq \mathsf{SC}(\mathbf{w}, \alpha) < (k+1) \cdot \mathsf{OPT}(\mathbf{w}), k \in \mathbb{N}.$$

We give a lower bound on the number of links that are necessary for such a pure Nash equilibrium α. We then use this lower bound to prove an upper bound on k.

Denote by δ_j the total traffic on link $j \in [m]$, denote by $\mathcal{M}_0$ the set of links with latency at least $k \cdot \mathsf{OPT}(\mathbf{w})$, and let $\Delta_0 = \sum_{j \in \mathcal{M}_0} \delta_j$. Thus,

$$\Delta_0 \;=\; \sum_{j \in \mathcal{M}_0} \delta_j \geq k \cdot \mathsf{OPT}(\mathbf{w}) \cdot |\mathcal{M}_0|. \qquad (1)$$

We show by induction on $1 \leq l \leq k-1$ that for all l, there exists a set of links $\mathcal{M}_l$, $\mathcal{M}_l \cap (\mathcal{M}_0 \cup \ldots \cup \mathcal{M}_{l-1}) = \emptyset$, such that the cardinality of $\mathcal{M}_l$ is at least

$$|\mathcal{M}_l| \;\geq\; (k-1) \prod_{1 \leq j \leq l-1} (k-j) \cdot |\mathcal{M}_0|, \qquad (2)$$

for all $j \in \mathcal{M}_l$, the total traffic on link j is bounded by

$$\delta_j \;\geq\; (k-l) \cdot \mathsf{OPT}(\mathbf{w}) \quad \forall j \in \mathcal{M}_l, \qquad (3)$$

the total traffic on links in $\mathcal{M}_l$ is at least

$$\Delta_l \;\geq\; (k-1) \prod_{1 \leq j \leq l} (k-j) \cdot |\mathcal{M}_0| \cdot \mathsf{OPT}(\mathbf{w}), \qquad (4)$$

and there exist users, assigned to links in $\mathcal{M}_0 \cup \ldots \cup \mathcal{M}_l$, with total traffic at least

$$\sum_{0 \leq i \leq l} (\Delta_i - |\mathcal{M}_i| \cdot \mathsf{OPT}(\mathbf{w})) \;\geq\; (k-1) \prod_{1 \leq j \leq l} (k-j) \cdot |\mathcal{M}_0| \cdot \mathsf{OPT}(\mathbf{w}). \qquad (5)$$

containing links in their strategy sets not in $\mathcal{M}_0 \cup \cdots \cup \mathcal{M}_l$.
First let $l = 1$. Since $\delta_i \geq k \cdot \mathsf{OPT}(\mathbf{w})$ for all $i \in \mathcal{M}_0$, there exist users assigned to links in $\mathcal{M}_0$ with total traffic at least $(k-1) \cdot \mathsf{OPT}(\mathbf{w}) \cdot |\mathcal{M}_0|$ containing links not in $\mathcal{M}_0$ in their strategy sets. This holds since it is possible to assign all users to links with latency at most $\mathsf{OPT}(\mathbf{w})$. Denote by $\mathcal{M}_1$, $\mathcal{M}_1 \cap \mathcal{M}_0 = \emptyset$, the set of these links. It follows that

$$|\mathcal{M}_1| \;\geq\; (k-1) \cdot |\mathcal{M}_0|, \qquad (6)$$

proving (2). Since each user causes latency at most $\mathsf{OPT}(\mathbf{w})$ on each link, the Nash condition implies $\delta_i \leq \delta_j + \mathsf{OPT}(\mathbf{w})$ for all $i \in \mathcal{M}_0, \forall j \in \mathcal{M}_1$. Thus, for all links $j \in \mathcal{M}_1$ the total traffic on link j is bounded from below by

$$\delta_j \;\geq\; (k-1) \cdot \mathsf{OPT}(\mathbf{w}) \quad \forall j \in \mathcal{M}_1, \qquad (7)$$

proving (3), and therefore

$$\Delta_1 \;=\; \sum_{j \in \mathcal{M}_1} \delta_j \overset{(7)}{\geq} (k-1) \cdot |\mathcal{M}_1| \cdot \mathsf{OPT}(\mathbf{w}) \qquad (8)$$
$$\overset{(6)}{\geq} \; (k-1)^2 \cdot |\mathcal{M}_0| \cdot \mathsf{OPT}(\mathbf{w}),$$

proving (4). Moreover, by (1),(6) and (8),

$$\sum_{0 \leq i \leq 1} (\Delta_i - |\mathcal{M}_i| \cdot \mathsf{OPT}(\mathbf{w})) \;\geq\; (k-1)^2 \cdot |\mathcal{M}_0| \cdot \mathsf{OPT}(\mathbf{w}),$$

proving (5), and thus the claim holds for $l = 1$.
For the induction step, let $l \geq 2$. By induction hypothesis,

$$\sum_{0 \leq i \leq l-1} (\Delta_i - |\mathcal{M}_i| \cdot \mathsf{OPT}(\mathbf{w})) \;\overset{(5)}{\geq}\; (k-1) \prod_{1 \leq j \leq l-1} (k-j) \cdot |\mathcal{M}_0| \cdot \mathsf{OPT}(\mathbf{w}).$$

Thus, there exist users assigned to links in $\mathcal{M}_0 \cup \ldots \cup \mathcal{M}_{l-1}$ with total traffic at least $(k-1) \prod_{1 \leq j \leq l-1} (k-j) \cdot |\mathcal{M}_0| \cdot \mathsf{OPT}(\mathbf{w})$ containing links not in $\mathcal{M}_0 \cup \ldots \cup \mathcal{M}_{l-1}$ in their set of allowed links, since it is possible to assign all users to links with latency at most $\mathsf{OPT}(\mathbf{w})$. Denote by $\mathcal{M}_l$, $\mathcal{M}_l \cap (\mathcal{M}_0 \cup \ldots \cup \mathcal{M}_{l-1}) = \emptyset$, the set of these links. It follows that

$$|\mathcal{M}_l| \;\geq\; (k-1) \prod_{1 \leq j \leq l-1} (k-j) \cdot |\mathcal{M}_0|, \qquad (9)$$

proving (2). Since each user causes latency at most $\mathsf{OPT}(\mathbf{w})$ on each link, the Nash condition implies $\delta_i \leq \delta_j + \mathsf{OPT}(\mathbf{w})$ for all $i \in \mathcal{M}_0 \cup \ldots \cup \mathcal{M}_{l-1}$ and for all $j \in \mathcal{M}_l$. Thus, for all links $j \in \mathcal{M}_l$ the total traffic on link j is bounded by

$$\delta_j \;\geq\; (k-l) \cdot \mathsf{OPT}(\mathbf{w}) \quad \forall j \in \mathcal{M}_l, \qquad (10)$$

proving (3), and therefore

$$\Delta_l \;=\; \sum_{j \in \mathcal{M}_l} \delta_j \overset{(10)}{\geq} |\mathcal{M}_l| \cdot (k-l) \cdot \mathsf{OPT}(\mathbf{w}) \qquad (11)$$
$$\overset{(9)}{\geq} \; (k-1) \prod_{1 \leq j \leq l} (k-j) \cdot |\mathcal{M}_0| \cdot \mathsf{OPT}(\mathbf{w}),$$

proving (4). By induction hypothesis, (9) and (11),

$$\sum_{0 \leq i \leq l} (\Delta_i - |\mathcal{M}_i| \cdot \mathsf{OPT}(\mathbf{w})) \;\geq\; (k-1) \prod_{1 \leq j \leq l} (k-j) \cdot |\mathcal{M}_0| \cdot \mathsf{OPT}(\mathbf{w}),$$

proving (5). This completes the proof of the inductive claim. We proceed by showing an upper bound on k. Since $m \geq |\mathcal{M}_{k-1}| + |\mathcal{M}_{k-2}|$, we get

$$m \;\geq\; |\mathcal{M}_{k-1}| + |\mathcal{M}_{k-2}| \overset{(2)}{\geq} |\mathcal{M}_0| \cdot k! = \Gamma(k+1).$$

Thus,

$$k \leq \Gamma^{-1}(m) - 1 = O\left(\frac{\log m}{\log \log m}\right).$$

Combined with Lemma 5.1, this proves the claim. $\square$

THEOREM 5.3. *Consider the model of identical users with restricted strategy sets on related links. Then, for any pure Nash equilibrium α, we have the tight bound*

$$\frac{\mathsf{SC}(\mathbf{w}, \alpha)}{\mathsf{OPT}(\mathbf{w})} = O\left(\frac{\log n}{\log \log n}\right).$$

PROOF. Since we consider identical users, the total traffic of a link $j \in [m]$ is the number of users assigned to j, divided by c_j. Similar to the proof of Theorem 5.2 we prove a lower bound on the number of users needed to construct a Nash equilibrium α with $\mathsf{SC}(\mathbf{w}, \alpha)/\mathsf{OPT}(\mathbf{w}) = k$. In the same way as in Theorem 5.2, this yields the stated upper bound. The tightness follows with Lemma 5.1 when $n = m$. $\square$

THEOREM 5.4. *Consider arbitrary users with restricted strategy sets on related links. Then, for any pure Nash equilibrium α, we have the tight bound*

$$\frac{\mathsf{SC}(\mathbf{w}, \alpha)}{\mathsf{OPT}(\mathbf{w})} \leq m - 1.$$

PROOF. Consider a pure Nash equilibrium α with $(k+1) \cdot \mathsf{OPT}(\mathbf{w}) > \mathsf{SC}(\mathbf{w}, \alpha) \geq k \cdot \mathsf{OPT}(\mathbf{w})$, $k \in \mathbb{N}$.
Upper bound: We now show by induction on $1 \leq i \leq k$ that there exist links $l_1, \ldots, l_i$ with latency $\lambda_{l_i} \geq (k - i + 1) \cdot$

$\mathsf{OPT}(\mathbf{w})$. Since $\mathsf{SC}(\mathbf{w},\alpha) = k \cdot \mathsf{OPT}(\mathbf{w})$, there exists a link $l_1 \in [m]$ with latency $\lambda_{l_1} \geq k \cdot \mathsf{OPT}(\mathbf{w})$, proving the claim for $i = 1$. Now assume $i \geq 2$. By induction hypothesis we know that there exist $i-1$ links $l_1, \ldots, l_{i-1}$ with $\lambda_{l_j} \geq (k-j+1) \cdot \mathsf{OPT}(\mathbf{w}) > \mathsf{OPT}(\mathbf{w})$ for all $1 \leq j \leq i-1$. Thus, there exists a user on $l_1 \cup \ldots \cup l_{i-1}$ that is assigned to some other link l_i in an optimal assignment, and the latency on l_i is at least $\lambda_{l_i} = (k-i+1)$ due to the Nash condition. This completes the proof of the inductive claim.
Since there exists at least one additional link with latency smaller than $\mathsf{OPT}(\mathbf{w})$, we have $m \geq k+1$ and therefore $k \leq m-1$. This proves the upper bound.
Tightness: Consider the following instance: We are given m links with capacities

$$c_j = \frac{(m-1)!}{(j-1)!} \quad \text{for all} \quad j \in [m],$$

and $m-1$ users with traffics $w_i = c_i$ and strategy set $\{i, i+1\}$ for all $i \in [m-1]$. The assignment α is defined as follows: We assign user i to link $i+1$ for all $i \in [m-1]$. Note that each user $i \in [m-1] \setminus \{1\}$ experiences latency $\frac{w_i}{c_{i+1}} = i$ on its link $i+1$, and that moving to the other link i in its strategy set would lead to latency

$$\frac{w_{i-1} + w_i}{c_i} \;=\; \frac{1}{c_i}\left(\frac{(m-1)!}{(i-2)!} + \frac{(m-1)!}{(i-1)!}\right) = i.$$

Furthermore, since $c_1 = c_2$, user 1 has no incentive to move from link 2 to link 1, showing that the given assignment is a Nash equilibrium. In an optimal assignment, each user $i \in [n]$ chooses link i as its strategy, yielding social cost $\mathsf{OPT}(\mathbf{w}) = 1$. This implies

$$\frac{\mathsf{SC}(\mathbf{w},\alpha)}{\mathsf{OPT}(\mathbf{w})} = m - 1,$$

and we are done. $\square$

6. REFERENCES

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows: theory, algorithms, and applications.* Prentice Hall, 1993.

[2] A. Czumaj and B. Vöcking. Tight bounds for worst-case equilibria. In *Proc. of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'02)*, pages 413–420, 2002.

[3] Y. Dinitz, N. Garg, and M.X. Goemans. On the single-source unsplittable flow problem. In *Proc. of the 39th Annual Symposium on Foundations of Computer Science (FOCS'98)*, pages 290–299, 1998.

[4] R. Feldmann, M. Gairing, T. Lücking, B. Monien, and M. Rode. Nashification and the coordination ratio for a selfish routing game. In *Proc. of the 30th Int. Colloquium on Automata, Languages, and Programming (ICALP'03)*, LNCS 2719, pages 514–526, 2003.

[5] R. Feldmann, M. Gairing, T. Lücking, B. Monien, and M. Rode. Selfish routing in non-cooperative networks: A survey. In *Proc. of the 28th Int. Symposium on Mathematical Foundations of Computer Science (MFCS'03)*, LNCS 2747, pages 21–45, 2003.

[6] D. Fotakis, S. Kontogiannis, E. Koutsoupias, M. Mavronicolas, and P. Spirakis. The structure and complexity of nash equilibria for a selfish routing game. In *Proc. of the 29th Int. Colloquium on Automata, Languages, and Programming (ICALP'02)*, LNCS 2380, pages 123–134, 2002.

[7] M. Gairing, T. Lücking, M. Mavronicolas, B. Monien, and P. Spirakis. Extreme nash equilibria. In *Proc. of the 8th Italian Conference on Theoretical Computer Science (ICTCS'03)*, LNCS 2841, pages 1–20, 2003, also accepted to *Theoretical Computer Science.*

[8] A.V. Goldberg. Efficient graph algorithms for sequential and parallel computers. PhD Thesis, Department of Electrical Engineering and Computer Science, MIT, 1987.

[9] A.V. Goldberg and S. Rao. Beyond the flow decomposition barrier. *Journal of the ACM*, 45:783–797, 1998.

[10] A.V. Goldberg and R.E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988.

[11] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.

[12] E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM*, 23(2):317–327, 1976.

[13] J. Kleinberg. Single-source unsplittable flow. In *Proc. of the 37th Annual Symposium on Foundations of Computer Science (FOCS'96)*, pages 68–77, 1996.

[14] S.G. Kolliopoulos and C. Stein. Improved approximation algorithms for unsplittable flow problems. In *Proc. of the 38th Annual Symposium on Foundations of Computer Science (FOCS'97)*, pages 426–435, 1997.

[15] S.G. Kolliopoulos and C. Stein. Approximation algorithms for single-source unsplittable flow. *SIAM Journal on Computing*, 31:919–946, 2002.

[16] E. Koutsoupias, M. Mavronicolas, and P. Spirakis. Approximate equilibria and ball fusion. *Theory of Computing Systems*, 36(6):683–693, 2003.

[17] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *Proc. of the 16th Int. Symposium on Theoretical Aspects of Computer Science (STACS'99)*, LNCS 1563, pages 404–413, 1999.

[18] J.K. Lenstra, D.B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

[19] T. Lücking, M. Mavronicolas, B. Monien, M. Rode, P. Spirakis, and I. Vrto. Which is the worst-case nash equilibrium? In *Proc. of the 28th Int. Symposium on Mathematical Foundations of Computer Science (MFCS'03)*, LNCS 2747, pages 551–561, 2003.

[20] R.D. McKelvey and A. McLennan. Computation of equilibria in finite games. In H. Amman, D. Kendrick, and J. Rust, editors, *Handbook of Computational Economics*, 1996.

[21] J. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951.

[22] C.H. Papadimitriou. Algorithms, games, and the internet. In *Proc. of the 33rd Annual ACM Symposium on Theory of Computing (STOC'01)*, pages 749–753, 2001.