Werth, T. L.; Holzhauser, M.; Krumke, Sven O.

Article

Atomic routing in a deterministic queuing model

Operations Research Perspectives

Provided in Cooperation with:
Elsevier

Suggested Citation: Werth, T. L.; Holzhauser, M.; Krumke, Sven O. (2014) : Atomic routing in a deterministic queuing model, Operations Research Perspectives, ISSN 2214-7160, Elsevier, Amsterdam, Vol. 1, Iss. 1, pp. 18-41, https://doi.org/10.1016/j.orp.2014.05.001

This Version is available at:
https://hdl.handle.net/10419/178243

Operations Research Perspectives 1 (2014) 18–41

Contents lists available at ScienceDirect

Operations Research Perspectives

journal homepage: www.elsevier.com/locate/orp

Atomic routing in a deterministic queuing model

T.L. Werth *, M. Holzhauser, S.O. Krumke

Technische Universität Kaiserslautern, Kaiserslautern, Germany

HIGHLIGHTS

- New model for atomic dynamic routing games (bottleneck and makespan/sum objective).
- Polynomial time algorithms for computing maximum and quickest dynamic flows.
- Greedy-type methods compute Nash equilibria in the dynamic game with sum objective.
- Dynamic minimum bottleneck flows and "narrowest paths" are NP-hard to compute.
- Nash equilibria for bottleneck objective may not exist, test and decision are NP-hard.

ARTICLE INFO

Article history:
Received 12 March 2014
Received in revised form
13 May 2014
Accepted 13 May 2014
Available online 2 June 2014

Keywords:
(I) Game theory
(B) Routing
Dynamic atomic unsplittable flows
Equilibria
Optimal solutions
Complexity

ABSTRACT

The issue of selfish routing through a network has received a lot of attention in recent years. We study an atomic *dynamic* routing scenario, where players allocate resources with load dependent costs only for some limited time.

Our paper introduces a natural discrete version of the deterministic queuing model introduced by Koch and Skutella (2011). In this model the time a user needs to traverse an edge e is given by a constant travel time and the waiting time in a queue at the end of e. At each discrete time step the first u_e users of the queue proceed to the end vertex of e, where u_e denotes the capacity of the edge e. An important aspect of this model is that it ensures the FIFO property.

We study the complexity of central algorithmic questions for this model such as determining an optimal flow in an empty network, an optimal path in a congested network or a maximum dynamic flow and the question whether a given flow is a Nash equilibrium.

For the bottleneck case, where the cost of each user is the travel time of the slowest edge on her path, the main results here are mostly bad news. Computing social optima and Nash equilibria turns out to be NP-complete and the Price of Anarchy is given by the number of users.

We also consider the makespan objective (arrival time of the last user) and show that optimal solutions and Nash equilibria in these games, where every user selfishly tries to minimize her travel time, can be found efficiently.

1. Introduction

Routing by independent users in a network can be viewed as a non-cooperative game, where selfish players choose their routes through the network (see e.g. [1,2]). From the game-theoretic point of view one can expect the routes chosen by the users to form a *Nash equilibrium* [3] or a strong equilibrium [4], which, in general, does not constitute an overall optimal solution (see [5]). The highest ratio of the social objective of a (strong) Nash equilibrium to the optimal objective is called (strong) Price of Anarchy and gives a measure for the maximum lack of performance that arises due to selfish behaviour (see [6–8]).

The issue of *selfish routing* has received a lot of attention in the recent years, from both the theory as well as the networking communities. Most of the work in the literature has been on static routing, where a user induces load on all edges of her path simultaneously. Recently dynamic models have gained more attention (see [9,10]). Here, each user travels through the network, and at any point in time she only adds load to the edge she is currently using.

We consider a natural discrete version of the deterministic queuing model introduced by Koch and Skutella [11]. Here, each

* Correspondence to: D-67663 Kaiserslautern, Paul-Ehrlich-Str. 14, Germany. Tel.: +49 631 205 4588.
E-mail addresses: werth@mathematik.uni-kl.de (T.L. Werth), holzhauser@mathematik.uni-kl.de (M. Holzhauser), krumke@mathematik.uni-kl.de (S.O. Krumke).

http://dx.doi.org/10.1016/j.orp.2014.05.001

edge has a constant travel time and a capacity that states the number of users who can leave an edge per time unit, while the others have to wait until the next point in time. The users leave the edge in the same order as they entered it (FIFO property). In the case that multiple users enter the edge at the same time different tie breaking rules are discussed. The goal is to find a flow with minimum travel time. In a game-theoretic context of a dynamic congestion game, each user chooses a path s.t. her personal travel time is minimized.

Most of our work is on bottleneck problems where the cost of a user is the most expensive resource on her path. There are several applications which motivate this model in the dynamic setting (for more applications of bottleneck games we refer to Banner and Orda [12]).

In the transport of living cargo (livestock) within the European Union there are numerous regulations regarding travel times and resting periods, see [13]. In particular, there are upper limits for the travel times between two stops. At every stop, livestock need to be fed/watered and inspected. In this setting, the links of the network do not correspond to physical roads but rather to aggregated travel routes between two replenishing points, so that a feeding/watering stop is mandatory after the traversal of every arc. The total time spent on a link, i.e., the constant travel time plus the time in the waiting queue, reflects the (aggregated) travel time, where the waiting time in a queue is due to congestion, e.g. at loading docks. Different types of vehicles have different effects on the congestion. Since for instance in extreme weather livestock suffer during transport and waiting, the bottleneck value on a path through the network can be viewed as an indicator of the risk for the animals. Due to the mandatory stops after every link, in fact the maximum travel time of a link is more appropriate here than the total travel time. Even though the links correspond to aggregated routes, still typical rules of traffic networks such as regulations about the right of way apply (e.g. major roads/minor roads, left-before-right-rule). This motivates the study of *local tie-breaking rules* when users enter a link simultaneously.

A second application occurs in routing in all-optical networks. Whenever more than one data packet arrives at a network node at the same time and multiple packets are destined for the same output, blocking occurs. In general, packets are processed in a first-in-first-out (FIFO) matter at every node, but there is also local priority ordering of the input ports associated with the node. Instead of dropping all but one packet, the current state-of-the-art technology uses fibre delay lines (FDLs) to delay the light, where packets circle until they can be forwarded, see e.g. [14–16]. For more information about all-optical networks we refer to standard textbooks such as [17]. There is a natural correspondence to waiting in an FDL and users spending time in the waiting queue in our model. The optical information deteriorates both on the links and in the fibre delay lines, but can be regenerated when the packet is finally forwarded. Since optical regeneration components are expensive (and somewhat slow), the FDLs are typically not equipped with this technology. The overall goal of a "good" routing is to minimize the maximum time between two nodes, the bottleneck value.

A related application occurs in (not necessarily all-optical) communication networks. As mentioned in [12], due to limited size of transmission buffers, there is an interest for the users who are sending data through the network to minimize the utilization of the most utilized buffer in order to avoid deadlocks and reduce packet loss. While Banner and Orda [12] considered the situation that a user permanently uses capacity on her chosen path, we consider a dynamic setting where this capacity is released once the data has left the link.

In a game-theoretic context, each message is sent by a selfish user who attempts to minimize the bottleneck value only on her path. If information from different edges enter a single node and a single edge with low capacity afterwards tie-breaking rules are needed to decide which user's information is processed first. In the networks considered here, the purifier acts on the information from one channel after the other and the order of the messages in one channel remains as it was before.

For other applications of (static) bottleneck games we refer to [12].

1.1. Related work

A deterministic queuing model in the non-atomic case was investigated by Koch and Skutella [11]. Here, each edge has a constant travel time and a capacity which states the amount of flow that can leave an edge per time unit. They show that Nash flows in dynamic congestion games can be computed as special static flows and give results about the Price of Anarchy. Anshelevich and Ukkusuri [18] investigate another non-atomic dynamic routing model, where each edge has a flow dependent latency function. They show that in the single-commodity case there is always a Nash equilibrium which can be computed efficiently.

Atomic models with flow dependent latency functions have been investigated by Farzad et al. [19] and Hoefer et al. [20]. In the first paper different models for the priorities of the users are discussed and the Price of Anarchy for (weighted) atomic models is computed. The second paper deals with the realistic effect that users delay other users after them. Therefore they introduce different local scheduling policies on the edges and show that the existence and computability of equilibria depends on the policy. For example, they show that the existence of equilibria is not guaranteed for every policy. Even for those, where the computation can always be done in polynomial time, a best-response step may be NP-hard to compute.

Work stated above deals with sum objective of the users. We mainly consider the bottleneck objective (see e.g., [12]). In the static case (even weighted) bottleneck games always have an optimal strong equilibrium, since a potential can be constructed from the bottleneck values of the players (see e.g. [21,22]). Furthermore, a Nash equilibrium can be computed in polynomial time in many cases. This can be done with a transformation to a minimum cost flow computation w.r.t. effective costs (see [23]) or by reducing the capacity on the resources with the help of an oracle (e.g., for network games it computes a maximum flow, see [24]). On the other hand, computing an optimal Nash equilibrium is NP-complete in multi-commodity networks (see [12]) or for weighted games (see [25]). The (strong) Price of Anarchy in these games has been investigated by many authors, e.g. by Banner and Orda [12]; Correa et al. [26]; Busch and Magdon-Ismail [27]; and Werth et al. [28]. In [12] it is shown that for linear latency functions the value is bounded from above by the number of edges and that this bound is tight, while for non-linear functions the value is unbounded. In [26] the fairness, i.e., the ratio of the best and the worst objective of a user, is computed, both for congestion and bottleneck games. Busch and Magdon-Ismail [27] showed that the Price of Anarchy is given by the length of a longest path in the network and in [28] it is shown that the strong Price of Anarchy in the unweighted case is given by 2.

1.2. Our contribution

We discuss an atomic, dynamic routing scenario in the deterministic queuing model for sum and bottleneck objectives. For the bottleneck objective we show that deciding whether a flow or path with a desired bottleneck value exists is NP-hard for many classes of instances (even for a single-commodity network with unweighted users). Furthermore, there are instances without a

Table 1
Overview on most complexity results in this paper. The first column states the problem, the second and third the assumptions for the positive (polynomial time solvability) and negative (NP-hardness) results in the sum case and the fourth column the assumptions for the negative results in the bottleneck case.

	Sum		Bottleneck
	Polynomially solv.	NP-hard	NP-hard
Path	–	–	Acyclic graph (Section 4)
Flow	Single-commodity, general graph (Section 6.1)	Two-commodity, acyclic graph (Appendix B.2)	Two-commodity, acyclic graph (Section 3)
Nash test		Two-commodity, acyclic graph (as bottleneck case)	Two-commodity, acyclic graph (Section 5.1)
Nash existence	Single-commodity, general graph (see Quickest Flow)	Multi-commodity, acyclic graph (Appendix B.3)	Multi-commodity, acyclic graph (Section 5.2)

Nash equilibrium and the question whether a given strategy profile is a Nash equilibrium or the question whether there is a Nash equilibrium for a given instance are both NP-hard. Furthermore, we give tight bounds on the (strong) Price of Anarchy for games with bottleneck objective. Specifically, we show that the (strong) Price of Anarchy and the Price of Anarchy are both equal to the number of users. This holds for unweighted and also weighted games, single commodities and acyclic graphs.

We also discuss the sum objective in a single-commodity network with unweighted users. Here, the computation of a maximum flow for a given time horizon and, hence the computation of a flow with minimum travel time can be done in polynomial time. Furthermore, a Nash equilibrium can also be computed efficiently, even in the weighted case under additional constraints (see Table 1).

Our paper is organized as follows. In Section 2 we formally introduce the deterministic queuing model and show that there are instances without a Nash equilibrium. Sections 3–5 consider the bottleneck case. Section 3 addresses the complexity of socially optimal flows with bottleneck objective. In Section 4 we prove that computing a narrowest path, i.e., a path with minimum bottleneck value in a network where there are already some units of flow routed on given paths, is NP-hard. The construction is used in the subsequent Section 5 to derive results about the hardness of testing a given strategy profile for being a Nash equilibrium. This section also contains a hardness result for the question whether a Nash equilibrium exists for a given instance and our tight bounds on the price of anarchy in dynamic games with bottleneck objective.

In Sections 6 and 7 we investigate the case where the personal cost of each user is the total length of her path. The main result of Section 6 is that a socially optimal flow with respect to the makespan objective can be computed in polynomial time. The efficient computation of equilibria in these games is addressed in Section 7.

2. Deterministic queuing model

The underlying model is similar to the static atomic routing scenario. An instance in the most general form is given by $\Gamma = [G = (V, E),\ (\tau_e,\ u_e)_{e\in E},\ \left(w_j\right)_{j=1,\dots,k},\ (s_j,\ t_j)_{j=1,\dots,k}]$.

Here, E is the set of m edges of a directed graph $G = (V, E)$, V is the set of n vertices, k is the number of unsplittable weighted users, w_j is the weight of user $j \in \{1, \dots, k\}$, τ_e is the constant free travel time and u_e is the capacity of edge $e \in E$. If $w_j = 1$ for all users j, only the number of users k is stated and the instance is called *unweighted* in that case. If $s_j = s$ and $t_j = t$ for all j, it is called a *single-commodity* instance.

We call a path P from s_j to t_j feasible for user j if it is simple and $u_e \geq w_j$ for all $e \in P$, i.e., the capacity of each edge on the path is at least the weight of the user. By $\mathscr{P}_j$ we denote the feasible paths for player j (also called her strategies) and by $\mathscr{P} = \bigcup_{j=1}^{k} \mathscr{P}_j$ their union.

A *strategy profile* $(P_j)_{j=1,\dots,k}$ specifies for every user j one feasible path $P_j \in \mathscr{P}_j$ the user allocates her weight to. We also call such a strategy profile a *(dynamic) flow* and write $f = \left(f_{P_j}\right)_{j=1,\dots,k}$ with $f_{P_j} = w_j$. While it may seem that specifying a flow by explicitly giving the path for each of the k users unnecessarily increases the input size, there are examples showing that this is inevitable in the dynamic model: different path decompositions of a flow specified only on the edges of the network may yield equilibria or not, depending on the particular decomposition.

In this paper we introduce an atomic variant of the deterministic queuing model by Koch and Skutella [11]: a user $j \in U$ who is in a vertex u at time t enters the next edge $e = (v, w)$ in her path immediately. After the free travel time τ_e she arrives at the end of the edge and is added to a waiting queue Q_e. This is a sorted list of the weights of the users who arrive at the end of edge e up to time $t+\tau_e$, but do not leave the edge to the next vertex before time $t+\tau_e$. They are stored in the queue Q_e in the same order as they entered it, i.e., if user i entered edge e before user j, then the entry for user i is ranked higher in the list Q_e than the entry of user j. In the case of simultaneous arrival we introduce priorities to break the tie (see the paragraph after the travel times).

Every step in time users with a total weight up to the capacity u_e of the edge e leave the edge to its end vertex w in the same order as they have been placed in the queue Q_e before. Let the entries (weights of the players) at time $t+\tau_e$ be given by $Q_e = (q_1, \dots, q_p)$ and $\alpha_1 \leq p$ maximum with $\sum_{i=1}^{\alpha_1} q_i \leq u_e$. Then the user corresponding to the first α_1 entries from Q_e leaves the queue, and hence, also the edge, at time $t + \tau_e$ without further waiting. By removing their entries from the list Q_e and repeating the computation, the time the other users leave the edge can be computed as well. Let $\alpha_l \leq p$ be maximum with $\sum_{i=\alpha_{l-1}+1}^{\alpha_l} q_i \leq u_e$ for $l \geq 2$. Note that we set $\alpha_0 = 0$ here. The computation stops when all users from the queue have been considered with an index r with $\alpha_r = p$. Let user j correspond to an entry in the set $\{\alpha_{l-1} + 1, \dots, \alpha_l\}$ for some $l \in \{1, \dots, p\}$. Then, she leaves edge e at time $t + \tau_e + l - 1$. The waiting time of user j on edge e in the flow f is denoted by $q_{e,j}(f)$. Since she entered edge e at time t and the waiting queue at time $t + \tau_e$, her waiting time is given by $q_{e,j}(f) = l - 1$. In the unweighted case the situation is even easier. Since each weight is equal to 1, the α_l are given by integer multiples of the capacity u_e, i.e., $\alpha_l = l\,u_e$. Then the waiting time of user j is given by $q_{e,j}(f) = \left\lfloor \frac{\pi_j - 1}{u_e} \right\rfloor$.

The latency (or travel time) of user j on edge e is defined as the free travel time plus the waiting time, i.e., it is given by $\ell_{e,j}(f) = \tau_e + q_{e,j}(f)$.

We assume that the free travel times τ_e are integral. This assumption is needed for games with bottleneck objective, since otherwise equilibria may not exist even in simple extension-parallel graphs. Furthermore, waiting in the vertices is not allowed in this model as discussed in the Introduction.

While in the non-atomic case the above conditions are enough to characterize the model (see [11]), the situation in the atomic case is more involved. If two users arrive at an edge at the same time it is not a priori clear in which order they enter the edge. This is in particular important if the capacity of the edge is smaller than

the sum of the weights of the users entering it. In order to resolve this issue we discuss two tie-breaking rules in this paper. In case of a tie, the user with better priority is placed on a higher position in the waiting queue. Note that better priority always means a smaller number for the priority.

The first rule assigns a global priority $\pi'_j(e) = j$ to each user $j \in \{1, \dots, k\}$ on every edge $e \in P_j$. In case of simultaneous arrival the user with better priority is always processed first. This rule is the natural choice used for congestion games (see the literature, e.g., [19]).

The other rule assigns a priority to each edge entering a vertex and, in the case of simultaneous arrival, the users from the edge with better priority are processed before the other users. The ordering between the users on the same edge is carried over from the previous edge they traversed, and, at a source vertex, which lacks ingoing edges, the users receive a starting priority. More precisely, for all edges e entering a vertex v we have a local priority $\pi'(e)$. Additionally, we introduce a starting priority $\pi'_j = j$ to all users $j \in U$ for the source vertex. With these two values the current priority $\pi'_j(e)$ for user j on edge $e \in P_j$ can be computed. Consider two users i and j who enter an edge e at the same time. Then the current priority $\pi'_i(e)$ of user i is better than the current priority $\pi'_j(e)$ of user j iff one of the following cases holds true:

(1) both users enter edge e after leaving different edges e_i and e_j, and the priority of the edge e_i of user i is better than the priority of the edge e_j of user j, i.e., $\pi'(e_i) < \pi'(e_j)$.
(2) both users enter edge e from the same edge e' and user i entered that edge before user j or, if they entered the edge e' at the same time, the priority of user i on that edge is better than the priority of user j, i.e., $\pi'_i(e') < \pi'_j(e')$.
(3) both users enter edge e from the source vertex s, and the starting priority of user i is better than the starting priority of user j, i.e., $\pi'_i < \pi'_j$.

We discuss two objectives in this paper. In a game with sum objective (as in congestion games) the personal cost user j wants to minimize is her total *travel time* $T_j(f) = \sum_{e \in P_j} \ell_{e,j}(f)$, i.e., the sum of the latency values of all edges in her path P_j. As a social objective we consider the makespan of the strategy profile, $T(f) = \max_{j=1,\dots,k} T_j(f)$, i.e., the highest travel time of all users. We refer to these games as dynamic routing games with sum objective in the deterministic queuing model and write in short as only game with sum objective.

In a game with bottleneck objective (as in bottleneck games) the personal cost user j wants to minimize is her *bottleneck value* $b_j(f) = \max\{\ell_{e,j}(f) \mid e \in P_j\}$, i.e., the highest latency value of all edges in her path P_j, while we choose the social objective to be the *bottleneck* of the game defined as $b(f) = \max_{j=1,\dots,k} b_j(f)$, i.e., the highest bottleneck value of all users. We refer to these games as dynamic routing games with bottleneck objective in the deterministic queuing model and write in short as only game with bottleneck objective.

If not stated otherwise, in games with sum objective the global tie-breaking rule is applied, which assigns priorities to the users. In games with bottleneck objective the local tie-breaking rule is applied, which assigns additional priorities to the edges. Most results for games with bottleneck objective hold true for the global tie-breaking rule, too. This is discussed in more detail later on.

We discuss two solution concepts in this paper, optimum solutions and equilibria. A strategy profile with minimal social objective is called an *optimal* strategy profile and the optimal objective value is denoted by $\mathrm{opt}(\Gamma)$.

A coalition of users $C \subseteq \{1, \dots, k\}$ is *satisfied* with their strategies, if not all of them can improve their personal objective values by changing their strategies simultaneously. When all single users, i.e., coalitions of size 1, are satisfied with their strategies, the strategy profile is called a *Nash equilibrium* (NE). If even all coalitions of users are satisfied with their strategies, then the strategy profile is called a *strong equilibrium* (SE).

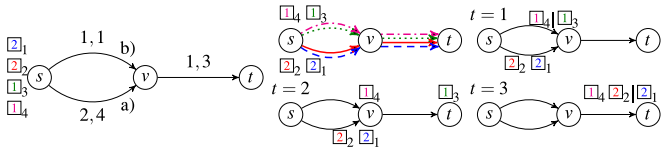


Fig. 1. Illustration of the model. Left: instance consisting of four weighted users. The weight is the boxed number and the index states the starting priority. The first number near an edge is the travel time and the second one the capacity. The expressions near the vertex v give the priority of the ingoing edge, where (a) is a better priority than (b). Right: the first picture (above left) shows the paths of the four users, the other pictures their positions in the network at different times t.

First we illustrate the deterministic queuing model with an example of a routing game with bottleneck objective.

Example 1. Consider the graph $G = (V, E)$ from Fig. 1 with $k = 4$ weighted users and the travel times and capacities as given near the edges in the picture. As for all games with bottleneck objective in this paper we use the local definition of the priorities for the users as defined above. To that end, the starting priorities of the users are given by the small indices to the weights of the users and the priorities of ingoing edges to vertex v are indicated with small letters (where (a) denotes the better priority than (b)). Note that the same instance without the priorities for the edges also defines an instance of a game with sum objective in our setting.

Users 1 and 2 with weight 2 travel along the lower path and users 3 and 4 with weight 1 along the upper path. At time 1 the users from the upper edge arrive at the end of the first edge. Since the capacity of that edge is 1, only one of the users is allowed to leave the edge to the vertex v. This is user 3, since she has a higher starting priority than user 4, who has to wait for the next point in time. More formally, the set of users in the waiting queue $Q = (1_3, 1_4)$ are separated in subsets of weight up to 1, i.e., in the two lists $Q_1 = (1_3)$ and $Q_2 = (1_4)$, where the users from the first one are allowed to leave the edge. The other two users are still travelling along the lower edge.

At time 2 user 3 already arrives at the terminal t and user 4 leaves her edge towards vertex v. Users 1 and 2 arrive at the end of the lower edge and line up in the queue. Here, user 1 is placed before user 2, since she had a higher priority at the source vertex. Then, both users leave the edge towards vertex v, since the capacity of the edge is large enough for both users. So, users 1, 2 and 4 enter the last edge at the same time and traverse it with travel time 1.

At time 3 the users arrive at the end of the edge and line up in the waiting queue. Since the lower edge has a better priority than the upper edge entering vertex v, the users from the lower edge are placed before user 4 in the queue. Furthermore, user 1 is placed before user 2, since they entered from the same edge and user 1 was placed before user 2 on the previous edge. Since the capacity of the edge is 3 and overtaking is not allowed, the queue $Q = (2_1, 2_2, 1_4)$ is separated into $Q_1 = (2_1)$ and $Q_2 = (2_2, 1_4)$. So, user 1 arrives at the terminal at time 3 and users 2 and 4 have to wait until time 4.

Note that the same instance can also be investigated for a game with sum objective. Here, we always use another tie-breaking rule with global priorities for the users and ignore the priorities given at the end of the edges. For the priorities chosen in this example the resulting flow is exactly the same. But, if users 3 and 4 had the best starting priorities, then the last one of these two would enter the terminal before the two heavier users. ◁

In the next example we show that, in contrast to static games, the existence of a Nash equilibrium in a dynamic routing game with bottleneck objective is not ensured.

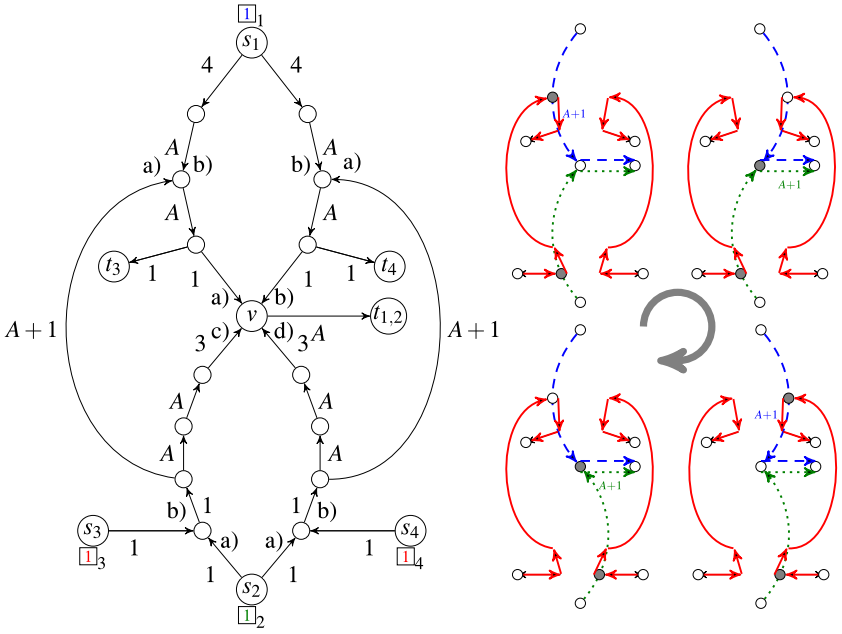


Fig. 2. No Nash equilibrium in an instance with unit capacities and a multi-commodity acyclic graph (on the left). Four relevant strategy profiles of the four users on the right. Here, dashed (blue) lines are used for the first commodity, dotted (green) lines for the second commodity and solid (red) lines for the other two; unused edges are omitted on the right. The numbers on some edges show the bottleneck values of the corresponding users from commodities 1 and 2 that cause them to be unsatisfied with their strategy. The small letters indicate the priorities of the ingoing edges for a vertex. The filled vertices show situations where users enter an edge at the same time. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

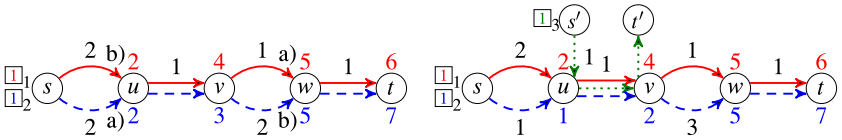


Fig. 3. Illustration of the model. The numbers near the edges state the travel times on the edges, the numbers near the vertices state the times the corresponding (main) user enters that vertex. Left: instance for local tie-breaking rules induced by the priorities of the edges. Here, the small letter states the priorities of the ingoing edges. Right: instance for global tie-breaking rules induced by the priorities of the users. Here is an additional auxiliary user on her s'–t'-path.

Example 2. Consider the graph $G = (V, E)$ from Fig. 2with $k = 4$ unweighted users and the travel times as given in the picture near the edges, where $A \geq 4$ as an instance of a game with bottleneck objective. All capacities are set to one and the priorities of ingoing edges are indicated with small letters. Commodities are denoted by (s_i, t_i) for user $i \in \{1, 2, 3, 4\}$ with priority i.

Note that users from commodities 3 and 4 do not have a choice on which route to take. Their purpose is slowing of the other users. So they are called blocking users. The users from commodities 1 and 2 have two choices each, because the outer paths have a bottleneck value of $A+1$, which is not acceptable as we see later on. Then there are four strategy profiles left that are drawn as graphs in the right of Fig. 2.

Whenever both (normal) users choose their left paths (upper left strategy profile), then the lower user arrives at her second vertex (filled) at the same time as the left blocking user. Since the path of the blocking users has an inferior priority, the blocking user is slowed down by one time unit. This is enough to reach the third vertex of the upper user at the same time as the user from commodity 1 does. Since the blocking users edge has a higher priority, the upper user has to wait one time unit, which results in a bottleneck value of $A + 1$ on that edge. Hence, she changes to the right path (upper right strategy profile). Here, the blocking user arrives after her at the third vertex and hence she arrives at vertex v with bottleneck value A. At this point her edge has a higher priority than the edge of the lower user who arrives at the same time. Hence, the lower user has to wait and gets a bottleneck value of $A + 1$. So, also the lower user changes her strategy to the right one (lower right strategy profile), which results in the same case as when both users took the left paths. Hence, there is no Nash equilibrium for this instance. ◁

The above example also shows that even for an instance with a single commodity a best-response dynamics may follow the cycle, and hence it is not guaranteed to find a Nash equilibrium in every case: if one adds a super source and a super terminal to the sources and sinks of the example respectively and one starts with the initial configuration of users as in one of the figures from 2, a best-response dynamics may follow the cycle and, hence, does not terminate.

A crucial point in the last example was the fact that one user could delay another user and be delayed herself by that user later on. This situation was caused by the fact that one time the priority of the edge from which the first user entered a vertex had higher priority than the edge from which the second user entered the same vertex and for another vertex the situation was the other way round. So, the question arises whether for global priorities of the users of such a situation can be avoided.

In the next example we illustrate that also for the tie-breaking rule with global priorities the same situation can occur. This can happen in the multi-commodity case or if a subset of the users routes on somehow fixed paths, since we need additional users to delay some of the main users.

Example 3. Consider the graphs from Fig. 3 with unweighted users and the travel times as given near the edges in the picture as an instance for a routing game. All capacities are set to 1. For the left

instance we use the local tie-breaking rule with priorities on the edges and for the right instance the global tie-breaking rule with global priorities for the users. For both instances we have two main users, and for the right one we have an additional auxiliary user.

In the left picture both main users arrive at vertex u at the same time 2. Since user 2 enters the vertex from an edge with better priority, she is placed before user 1 on the upcoming edge and hence also enters vertex v before user 1, since all capacities are set to 1. Then user 2 chooses the longer edge towards vertex w. Therefore, both users arrive at the end of vertex (w, t) at the same time. Since user 1 enters vertex w from an edge with better priority here, she is placed before user 2 on the upcoming edge and hence also enters the terminal t before user 2. In total, each user delays the other user once.

In the right picture there is an additional auxiliary user. Main user 2 arrives at vertex u at the same time (1) as the auxiliary user. Since she has the better priority, she is placed before the auxiliary user on the upcoming edge and hence also enters vertex v before the auxiliary user, who has to wait in the waiting queue until the next point in time. At time 2 main user 1 also enters the queue, but since she entered later than the auxiliary user she is placed in the queue after her (although she has a better priority). So, she arrives at vertex v at time 4. Afterwards, as in the left instance, both main users arrive at vertex w at the same time. Since main user 1 has a better priority than main user 2 in such a case, she is placed before main user 2 on the upcoming edge and hence also enters the terminal t before main user 2. In total, each of the main users delays the other main user once. ◁

Applying the same technique of adding additional users to the instance from Example 2 shows that the existence of a Nash equilibrium in a dynamic routing game with bottleneck objective is also not ensured for global priorities.

3. The Narrowest Flow problem

In this section we show that finding a flow with minimum bottleneck value, i.e., a social optimum in a game with bottleneck objective, is hard in general. This is shown for different scenarios: in the multi-commodity case, even if the graph is acyclic and the users are unweighted, in a general graph, even if it has only a single commodity and the users are unweighted and for weighted users, even on a graph consisting of parallel links.

Definition 1 (NarrowestFlow *Problem*)**.**

Input:	Instance $\Gamma = [G = (V, E),\ (\tau_e,\ u_e)_{e\in E},\ k,\ (s_j,\ t_j)_{j=1,\dots,k}]$ and a bound b.
Question:	Is there a feasible flow $f = \left(f_{P_j}\right)_{j=1,\dots,k}$ for that instance with bottleneck value smaller or equal to b?

Theorem 1. NarrowestFlow *is* NP*-complete in the strong sense in the two-commodity case, even if the graph is acyclic.*

Proof. The proof uses a reduction from the 3-SAT problem as in [29]. Recall that an instance of 3-SAT is given by a set of Boolean variables $X = \{x_0, \dots, x_{\eta-1}\}$, and ρ clauses $(C_j)_{j=1,\dots,\rho}$ containing exactly three literals (variables or their negations). The question is whether there is a truth assignment $T: X \to \{0, 1\}$ s.t. any clause C_j contains at least one literal which is true. In the sequel we denote by $\ell_i \in \{x_i, \bar{x}_i\}$ a literal involving the variable x_i, and we extend an arbitrary truth assignment $T: X \to \{0, 1\}$ in the standard way to the negated literals by setting $T(\bar{x}_i) := 0$ if $T(x_i) = 1$ and $T(\bar{x}_i) := 1$ if $T(x_i) = 0$.

We now construct an instance of NarrowestFlow in such a way that there is a flow with bottleneck value equal to 1 iff there is a Yes-answer for 3-SAT.

For each literal vertex x_i there is a gadget as illustrated in the left part of Fig. 4 containing of a source vertex v_s^i and a terminal vertex v_t^i, which are connected by two disjoint paths $(v_s^i, v_1^i, v_2^i, \dots, v_{2p_i}^i, v_t^i)$ and $(v_s^i, \overline{v}_1^i, \overline{v}_2^i, \dots, \overline{v}_{2p_i}^i, v_t^i)$. Here, p_i is the number of occurrences of the variable x_i in the clauses and q_i is the number of occurrences of the negated variable $\bar{x}_i$ in the clauses. We denote by $\theta = \max\{p_i, q_i\}$ and set $T = \eta\theta + 1$. The travel time from $v_{2p_i}^i$ to v_t^i is given by $\theta - 2p_i - 1$ and the travel time from $\overline{v}_{2q_i}^i$ to v_t^i is given by $\theta - 2q_i - 1$. All other travel times in these two paths and all capacities are set to 1.

Furthermore, there are two commodities: (s_1, t_1) with one user and (s_2, t_2) with ρ users. Their actual priorities are not important for this proof. The first source vertex s_1 is connected via an edge with travel time 1 to vertex v_s^0 in the first gadget (the one of variable x_0). The gadget of variable x_i is connected to the gadget of variable x_{i+1} for all $i \in \{0, \dots, \eta - 2\}$. This is done with an edge from vertex v_t^i in gadget i to vertex v_s^{i+1} in gadget $i+1$ with travel time 1. The last gadget is connected to the terminal with an edge from vertex $v_t^{\eta-1}$ to the terminal vertex t_1. In total we have $k = 1 + \rho$ unweighted users.

For each clause C_j there is a clause vertex C_j. From each of these vertices there is an edge to the second terminal vertex t_2. The second source vertex s_2 is connected to vertex v_{2l-1}^i for $l \in \{1, \dots, p_i\}$ in gadget $i \in \{0, \dots, \eta - 1\}$ with an edge with travel time $i\theta + l + 1$. The vertex v_{2l}^i for $l \in \{1, \dots, p_i\}$ in each gadget $i \in \{0, \dots, \eta - 1\}$ is connected to the lth clause vertex that contains the literal x_i via an edge with travel time $T - i\theta - l - 3$.

The actual priorities of the edges are also not important for the reduction, so they are chosen arbitrarily. Note that for the current proof we need to replace all edges with travel time greater than 1 by a path of consecutive edges with travel times equal to 1 and the same total length. It is not stated that way directly, since we will reuse the construction later on without this restriction for the makespan objective. The whole reduction is illustrated in the bottom part of Fig. 4.

Note that the first user arrives at vertex v_l^i at time $i\theta + l + 1$, if she is not slowed down by one of the other users before. Furthermore, a user from the second commodity traversing vertex v_l^i arrives there at the same time. Hence, if both users traverse the same vertex, one of them is slowed down by one time unit. In the absence of the users from the second commodity, the first user arrives at her terminal vertex t_1 at time $\eta\theta + 1 = T$. In the absence of the other users, a single user from the second commodity traversing a shortest path arrives at the clause vertex she traverses at time $i\theta + l + 1 + 1 + T - i\theta - l - 3 = T - 1$ and at her terminal vertex t_2 at time T.

In a flow with bottleneck value ≤ 1, the edges connecting the gadgets can only be used by the first user. Hence, any user from the second commodity can only traverse a single gadget. Since every clause is connected to the second terminal vertex t_2 by a single edge and a total amount of ρ users have to be routed for the second commodity, a flow with bottleneck value ≤ 1 has to route a single unit of flow along each clause vertex. Furthermore, no user in this flow can be slowed down by another user, since every path has capacity 1, free travel time T and all users traversing a subpath of the path arrive at the common vertices at the same time.

With these observations it is easy to show that there is a flow with bottleneck value equal to 1 iff there is a feasible truth assignment.

Let f be a feasible flow with bottleneck value 1. Then, a single unit of flow routes from s_1 to t_1 without meeting another user. We set the variable x_i to true if this unit routes in the ith gadget along the lower path and we set the variable x_j to false if it routes in the jth gadget along the upper path. Then the users from the second commodity route along the vertices from the upper path in gadget i and along vertices from the lower path in gadget j. From these

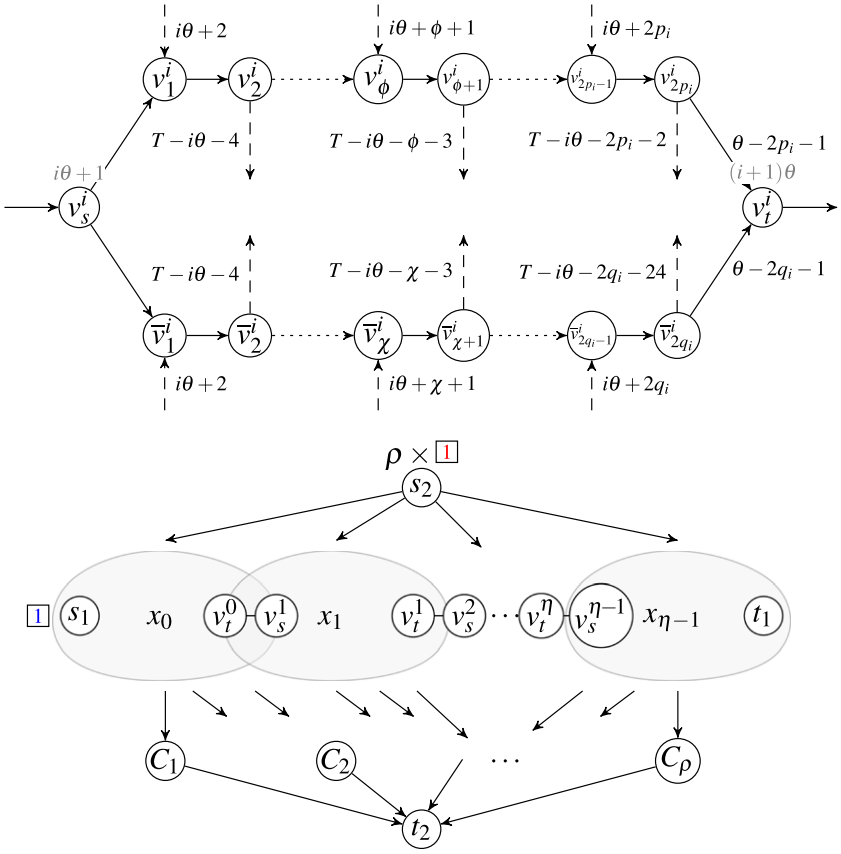


Fig. 4. Illustration of the reduction from 3-SAT to the Narrowest Flow problem in the two-commodity case. Top: the graph shows a gadget representing a literal $x_i, i \in \{0, \ldots, \eta-1\}$. The expressions near some of the edges are the travel times of these edges. The expressions near some of the vertices denote the expected arrival times at these vertices, if there is a feasible truth assignment. All capacities are set to 1, edge priorities are omitted. Bottom: shows the whole graph. The greyly shaded areas illustrate the gadgets.

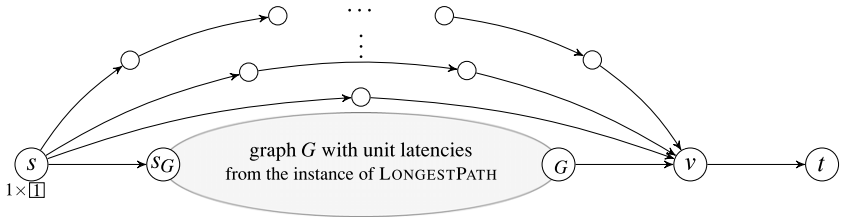


Fig. 5. Illustration of the reduction from LONGESTPATH in a graph G to dynamic NARROWESTFLOW in the drawn graph.

vertices one unit of flow is routed to each of the clause vertices and afterwards to the sink. Since at least one vertex in the path of a gadget of the corresponding variable is connected to a clause it is contained in any clause there is one vertex that is set to true by the flow. So, the flow induces a truth assignment.

On the other hand, given a truth assignment, we route the user from the first commodity in the ith gadget along the lower path if the variable x_i in the truth assignment is true and we route the user in the jth gadget along the upper path if the variable x_j is false. At least one vertex in the path of a gadget of the corresponding variable is connected to a clause it is contained in. Additionally, in any clause there is one variable that is set to true. Hence, for the second commodity we route one unit of flow towards each clause vertex to which no unit of flow from the second commodity was already routed to. It is sent on a shortest path along a vertex in the upper path of gadget i or in the lower path of gadget j (depending on the choice for the first commodity). This gives a feasible flow and no two units of flow use the same edge. Hence, the bottleneck value of this flow is given by 1. □

The above construction handles the multi-commodity case. Now we show that the problem remains NP-complete in the single-commodity case if we allow general graphs without restricting the problem to acyclic graphs.

Theorem 2. NARROWESTFLOW *is* NP*-complete in the strong sense in general graphs with unweighted users, even if there is only a single commodity.*

Proof. We provide a polynomial-time reduction from LONGESTPATH. Given an instance of LONGESTPATH, i.e., a graph $G = (V, E)$ with $|E| = m$, source–sink pair (s_G, t_G), latencies $\ell_e = 1$ for all $e \in E$ and a constant $K \leq m$, the question is whether there is a simple path of length exactly K. It is shown by Garey and Johnson [30] that LONGESTPATH (in this variant) is NP-complete in the strong sense.

For the instance of NARROWESTFLOW we construct a graph G' as shown in Fig. 5 and take $k = m+1$ users. There are m paths P_i from s to v with $i \in \{2, \ldots, m+2\} \setminus \{K+2\}$ edges in a row. All travel times and capacities are set to 1.

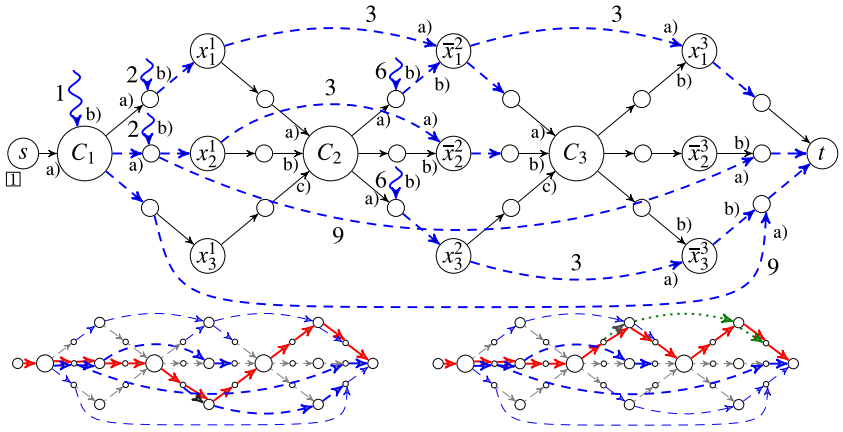


Fig. 6. Top: illustration of the reduction from 3-SAT to NarrowestPath. The clauses are $C_1 = (x_1 \vee x_2 \vee x_3)$, $C_2 = (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$ and $C_3 = (x_1 \vee \bar{x}_2 \vee \bar{x}_3)$ over the variables $\{x_1, x_2, x_3\}$. The free travel times $\neq 1$ are written near the edges, the priorities of ingoing edges are small letters. Below: a feasible truth assignment (left: $x_2 = x_3 = x_1 = 1$) and an infeasible one (right: $x_2 = \bar{x}_1 = x_1 = 1$) illustrated by solid (red) paths. The dotted (green) path shows where the bottleneck is increased to 2 for the infeasible assignment. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

We show that there is a flow with bottleneck value 1 in G' $\Longleftrightarrow$ there is a simple path with length exactly K in G.

$\Longleftarrow$: We send one user along the path that consists of edge (s, s_G), the simple path of length K in G and the part (t_g, v, t). The other m users are sent along the m disjoint upper paths and the edge (v, t). Since all of these m paths arrive at v at different times between 2 and $m+2$, but none at time $K+2$, there is no edge with load >1 and hence, no user receives a bottleneck value higher than 1.

$\Longrightarrow$: Given a flow with bottleneck value 1, there has to be exactly one user to traverse graph G. Furthermore, there cannot be two users on edge (v, t) at the same time. Hence, the user going through graph G has to arrive at vertex v at time $K+2$. So, she has to take a simple path of length exactly K in G. □

As in other dynamic models, the first non-trivial class of graphs are those consisting of parallel links and an additional edge afterwards with unit free travel times and capacities. One can show that an optimal solution there can be stated explicitly. Note that even in this graph the dynamic NarrowestFlow problem is NP-complete in the strong sense in the weighted case. This can be easily shown by a reduction from 3-Partition: an instance of 3-Partition is given by $3m$ integers a_i, $i \in \{1, \ldots, 3m\}$, which sum up to $m \cdot A$, where $A/4 < a_i < A/2$. The question is whether there are m subsets consisting of 3 integers each s.t. the sum of the integers in each subset is equal to A. Given such an instance of 3-Partition, we construct a graph consisting of m parallel edges with free travel time 1 and capacity A each. We also add $3m$ users with weights a_i, $i \in \{1, \ldots, 3m\}$. It is immediate that a flow has bottleneck value equal to 1 iff the sum of the weights of the users on each edge is exactly A, which induces a solution to 3-Partition. The reason is the following: there would be an edge with load lesser than A iff there would be an edge with load greater than A. Hence, in an optimum solution (with bottleneck value 1), each edge has load exactly A. Since the weights of the users are in $(A/4, A/2)$, there cannot be four or more users on an edge and there cannot be two or less users on an edge.

4. The narrowest path problem

In the previous section we showed that finding a dynamic flow with minimum bottleneck value is hard. In this section we show that finding a single path with minimum bottleneck value is already NP-hard, if there is already some flow routed in the network. This flow already routed before is the reason why the hardness result does not subsume Theorem 1. We need the current hardness of the path problem later on to show the hardness of testing whether a given strategy profile is a Nash equilibrium. Since this is shown even for the unweighted case, we also show the hardness result in this section for routing a single unit of flow along the additional path.

Definition 2 (NarrowestPath *Problem*)**.**

Input:	Instance $\Gamma = [G = (V, E),\ (\tau_e,\ u_e)_{e\in E},\ k,\ (s_j,\ t_j)_{j=1,\ldots,k}]$ of a game with bottleneck objective, flow $f = (f_{P_j})_{j=1,\ldots,k-1}$ for $k-1$ users and a bound b.
Question:	Is there a path P_k for user k s.t. the bottleneck value $b_k(\tilde{f})$ of that user in the flow $\tilde{f} = (f_{P_j})_{j=1,\ldots,k}$ with $f_{P_k} = 1$ is smaller or equal to b?

Note that the users already routed have a better priority than the new user k, who has weight 1.

Theorem 3. NarrowestPath *is* NP*-complete in the strong sense, even if the graph is acyclic and only a single unit of flow has to be routed.*

Proof. We reduce 3-SAT (see [30]) to NarrowestPath. Suppose that we are given an instance of 3-SAT. We now construct an instance of NarrowestPath in polynomial time in such a way that there is a path with bottleneck value equal to 1 iff there is a Yes-instance for 3-SAT (the construction is illustrated in Fig. 6).

Construction—clause vertices: For every clause C_j with literals $\ell_{j_1}, \ell_{j_2}, \ell_{j_3}$, we add a clause vertex C_j and three literal vertices $\ell^j_{j_1}, \ell^j_{j_2}, \ell^j_{j_3}$ (so in total, for every literal there will be as many literal vertices as this literal occurs in the clauses). We add a super source s and a super termination t. For the ease of notation, the terminal is also denoted by $C_{\rho+1}$. The source s is connected to the first clause vertex C_1 via a single edge with high priority.

Construction—literal vertices: For each literal vertex ℓ^j_i corresponding to a clause vertex C_j we add a path from the clause to the literal, i.e., $(C_j,\ v^j_{i,1}, \ldots, v^j_{i,\rho-2},\ \ell^j_i)$, consisting of $\rho-1$ consecutive edges with $\rho-2$ vertices $v^j_{i,l}$ with $l \in \{1, \ldots, \rho-2\}$ in between. We call these vertices preliteral vertices and set $v^j_{i,\rho-1} = \ell^j_i$ and $v^j_{i,0} = C_j$. The last edge entering ℓ^j_i has a low priority, while all other edges have high priority. The exact numbers are not important. Furthermore, there is a path $(\ell^j_i,\ w^j_{i,1}, \ldots,\ w^j_{i,\rho-2},\ C_{j+1})$ consisting of $\rho-1$ consecutive edges with $\rho-2$ vertices $w^j_{i,l}$ with

$l \in \{1, \ldots, \rho - 2\}$ in between. We call these vertices postliteral vertices and set $w^j_{i,0} = \ell^j_i$ and $w^j_{i,\rho-1} = C_{j+1}$. Note that each literal and each clause vertex belong to the pre- and postliteral vertices, too. The last edge entering C_{j+1} has a high priority, while all other edges have low priority. Again, the exact numbers are not important. All travel times are set to 1 unless stated differently (in the figure, these are the travel times not stated explicitly).

The distance from s to a preliteral vertex $v^j_{i,l}$ is given by $\tau\left(v^j_{i,l}\right) = (2j-2)(\rho-1)+1+l$ and the one from s to a postliteral vertex $w^j_{i,l}$ is given by $\tau\left(w^j_{i,l}\right) = (2j-1)(\rho-1)+1+l$. Hence, the distance from s to a clause vertex C_j is given by $\tau\left(C_j\right) = (2j-2)(\rho-1)+1$ and the distance from s to a literal vertex ℓ^j_i is given by $\tau\left(\ell^j_i\right) = (2j-1)(\rho-1)+1$.

Construction—bent edges: The important (additional) edges are those that forbid us using a literal and its negated one. They do not necessarily connect the two literal vertices directly, but they start at a preliteral and end at a postliteral in such a way that two of these edges do not enter at the same vertex. So, let ℓ^j_i be a literal vertex and $\ell^{j'}_{i'}$ be a literal vertex in a later clause, i.e., $j' > j$, corresponding to the literal that is the negation of the literal corresponding to the first vertex. We add an edge from the preliteral vertex $v^j_{i,\rho+j-j'}$ to the postliteral vertex $w^{j'}_{i',j'-j-1}$ with travel time $\tau\left(w^{j'}_{i,j'-j-1}\right) - \tau\left(v^j_{i,\rho+j-j'}\right) - 1 = 2(j'-j)\rho - 3$ and high priority. These edges are called bended edges (as they are drawn bent in the picture). Note that by traversing this edge, one arrives at the postliteral vertex $w^{j'}_{i,j'-j-1}$ one time unit earlier than by travelling through the network without these additional edges. The indices $l = \rho + j - j'$ and $l' = j' - j - 1$ are chosen in such a way that arcs towards a later clause (large j') start earlier (smaller l) and arcs from a prior clause (small j) enter later (larger l'). This gives a nicer picture, since arcs between two consecutive clause are given by $\left(\ell^j_i,\ \ell^{j+1}_{i'}\right)$ and do not need the auxiliary vertices v and w. Furthermore, we add an edge from s to the preliteral vertex $v^j_{i,\rho+j-j'-1}$ with travel time $\tau\left(v^j_{i,\rho+j-j'-1}\right) = (2j-2)(\rho-1)+\rho+j-j'$ and low priority together with an edge from the postliteral vertex $w^{j'}_{i,j'-j}$ to the termination t with travel time 2. Note that this is possible since $\rho + j - j' - 1 \in \{0, \ldots, \rho-2\}$ and $j' - j \in \{1, \ldots, \rho-1\}$. In the illustration, edges from the source are only indicated by squiggled solid lines (one of the two edges to C_1 is omitted), and all edges to the termination are omitted for optical reasons. These edges are called squiggled edges.

Construction—blocking users: For all pairs of literals and negated literals in later clauses, we route one user on path $\left(s,\ v^j_{i,\rho+j-j'-1},\ v^j_{i,\rho+j-j'},\ w^{j'}_{i',j'-j-1},\ w^{j'}_{i',j'-j},\ t\right)$ (dashed (blue) paths in the upper picture). These users are called blocking users and are needed to forbid the main user traversing a vertex corresponding to a literal and another vertex corresponding to its negated literal later on. In total there are not more than $3\rho^2$ users, and their paths do not intersect (on an edge) at all. The resulting graph is acyclic, and all capacities are set to 1.

Remarks. Before we prove that the reduction works, let us briefly deduce some structural properties about the s–t-path P of an additional user. This user is also called the main user. The graph was constructed in such a way that the main user receives a bottleneck value of at least 2, if she travels along one of the bended edges or travels along an edge at the same time as one of the blocking users, but with an inferior priority.

Claim 1. The main user arrives at vertex C_j at time $\tau\left(C_j\right) = (2j-2)(\rho-1)+1$ with a bottleneck value of 1 iff she does not travel along any of the bended or squiggled edges or along an edge as one of the blocking users at the same time with an inferior priority.

Proof. If the main user has a bottleneck value of 1, then she does not traverse the edges as claimed above. On the other hand, if she does not travel along these edges, her travel time on each edge in her path is equal to 1. Then she arrives at vertex C_j at time $\tau\left(C_j\right) = (2j-2)(\rho-1)+1$. ◁

Claim 2. If main user arrives at vertex C_j at time $\tau\left(C_j\right)$ with a bottleneck value of 1, then all blocking users are slowed down by one time unit on their first common edge with the main user.

Proof. Let the first edge blocking a user and the main user traverse at the same time be edge $e = (v^j_{i,\rho+j-j'-1}, v^j_{i,\rho+j-j'})$. Then the main user arrives at the starting vertex at time $\tau\left(v^j_{i,\rho+j-j'-1}\right) = (2j-2)(\rho-1)+\rho+j-j'$ and the blocking user arrives also at time $\tau\left(v^j_{i,\rho+j-j'-1}\right) = (2j-2)(\rho-1)+\rho+j-j'$. Since the edge of the main user has a better priority, the blocking user has to wait in the waiting queue for one time unit and hence, reaches the end vertex of the edge one time unit after the main user. ◁

Claim 3. If the main user traverses a literal vertex and a corresponding negated literal vertex afterwards, then her bottleneck value is at least 2.

Proof. Let the main user traverse vertex ℓ^j_i corresponding to a literal ℓ and vertex $\ell^{j'}_{i'}$ corresponding to the negated literal $\bar{\ell}$ afterwards in such a way that the travel time on each edge up to vertex $\ell^{j'}_{i'}$ is given by 1. Then there is a bended edge between a preliteral vertex $v^j_{i,\rho+j-j'}$ and a postliteral vertex $w^{j'}_{i',j'-j-1}$. Both vertices are traversed by the main user and a blocking user, who are slowed down by the main user by one time unit on the edge towards vertex $v^j_{i,\rho+j-j'}$, where she arrives one unit of time after the main user (see Claim 2). Since the travel time of the bended edge is one time unit less than the path along the clause vertices, the main user and the blocking user both arrive at vertex $w^{j'}_{i',j'-j-1}$ at time $\tau\left(w^{j'}_{i',j'-j-1}\right)$. Then the main user and the blocking user both traverse the edge $e' = \left(w^{j'}_{i',j'-j-1},\ w^{j'}_{i',j'-j}\right)$. Since the bended edge of the blocking users has a better priority than the edge of the main user, the travel time of the main user on edge e' is increased by one and hence, her bottleneck value cannot be 1 any more. ◁

Main Claim. There is a path with bottleneck value ≤ 1 iff there is a satisfying assignment for the instance of 3-SAT.

$\Longleftarrow$: Let T be a satisfying truth assignment. So, in any clause C_j there is at least one literal that is set to true. We will denote this literal by ℓ_{i_j} and the corresponding literal node by $\ell^j_{i_j}$. If there is more than one true literal in C_j, we pick an arbitrary one.

We route the new user from the source s to the first clause node C_1, from a clause node C_j to the literal node $\ell^j_{i_j}$ corresponding to the true literal ℓ_{i_j} in the clause C_j and then to the next clause node C_{j+1} up to the termination node $C_{\rho+1} = t$. This is a feasible path which by the analysis above has bottleneck value 1, since it traverses none of the bended edges and no literal vertex and a negated literal vertex later on.

$\Longrightarrow$: Assume conversely that there is a path P with bottleneck value 1 for the main user. Then the path cannot traverse any of the

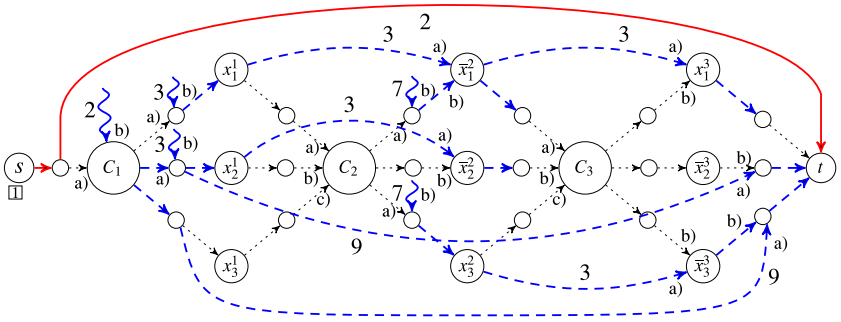


Fig. 7. Illustration of the modelling of 3-SAT for the reduction to NashTest. The numbers near the edges are the travel times. If there is no number near an arc, then its travel time is 1. The letters give the priority of ingoing arcs. The dashed (blue) paths belong to the blocking users, the solid (red) path to the main user and the dotted (black) paths are not used at all. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

bended edges with travel time greater than 1. Furthermore, path P cannot intersect with the same blocking user more than once by Claim 3. So the new path does not go from a literal node to a node of the negated literal. Then, the path implies a truth assignment for 3-SAT, since it traverses the node corresponding to one of the literals in each clause and never enters a node of a literal and one of its negated literal. □

Remark. Note that the reduction also works for the global tie-breaking rule, if the blocking users get best priorities, the main user gets a normal priority, and for each blocking user an additional auxiliary user is added with bad priority. Furthermore, the main user and the auxiliary users have to arrive at the vertices where the main user delayed the blocking users in the reduction from above at the same time and one time unit before the blocking user. If the travel times of the blocking users are decreased accordingly, then one can show that the blocking user is again delayed and the equivalence from above is still true. □

Note that in extension-parallel graphs [31] the narrowest path problem becomes polynomial-time solvable: there are only $\mathcal{O}(n)$ different paths from s to t and one can exhaustively search for the best one.

5. Nash equilibria in games with bottleneck objective

We have seen that there may be instances without a Nash equilibrium (Example 2). The first non-trivial class of graphs are those consisting of parallel links and an additional edge afterwards with unit free travel times and capacities. One can show that these always have a Nash equilibrium, which can be stated explicitly. On the other hand, easy examples show that even in extension-parallel graphs, no Greedy type algorithm is guaranteed to find a Nash equilibrium (if one exists). But, the situation is even worse in the dynamic setting: we show that the question whether a given strategy profile is a Nash equilibrium is already co-NP-complete, the question whether a given instance has a Nash equilibrium is NP-complete and the Price of Anarchy is very high.

5.1. Complexity of the dynamic NashTest problem

In this section we show that deciding whether a given strategy profile is a Nash equilibrium or not is co-NP-complete for multi-commodity instances, even for unweighted users.

Definition 3 (NashTest *Problem*)**.**

Input:	Instance $\Gamma = [G = (V, E),\ (\tau_e,\ u_e)_{e\in E},\ k,\ (s_j,\ t_j)_{j=1,\dots,k}]$ of a game with bottleneck objective and a flow $f = (f_{P_j})_{j=1,\dots,k}$ for k users.
Question:	Is f is a Nash equilibrium?

Corollary 4. NashTest *is co-NP-complete in the strong sense for multi-commodity instances, even if all users are unweighted and the graph is acyclic.*

Proof. The problem is in co-NP, since given a strategy profile and an alternative strategy for one of the users it is easy to see whether this strategy is better than the old one or not.

To show the hardness we reduce 3-SAT to NashTest. In order to do that we modify the reduction to NarrowestPath slightly (see Fig. 6) s.t. the blocking users do not have any intention to change their strategies:
We add an additional vertex h between the source s and the first clause vertex C_1 and two arcs (s, h) and (h, C_1) with travel time 1 together with an arc from h to t with travel time 2. Furthermore, all travel times of arcs leaving s added for the blocking users are increased by 1, too. Since in the NashTest problem also the blocking users are not satisfied with high bottleneck values, the travel time of any single edge has to be not higher than 2. To achieve that, all arcs with travel times ≥ 2 (those above and these from literal vertices to negated literal vertices added for the blocking users) are replaced by paths consisting of consecutive arcs, where at least one of the arcs in each of these paths has travel time 2 and the others have travel times 1 or 2 such that the travel time of the path is equal to the travel time of the old arc. The capacity of all arcs is equal to the capacity of the old arc. All other travel times between the drawn vertices, all capacities and priorities stay the same due to this change.

To complete the instance of NashTest we provide the flow that has to be tested. This flow is given as follows: the blocking users take the same paths as the blocking users in the NarrowestPath instance with the exception that they do not use the old single arcs, since these have been replaced by the new paths of consecutive arcs as discussed above. The paths are indicated by squiggled solid and dashed (blue) edges in the picture. Note that the edges towards the termination are omitted for optical reasons as for the NarrowestPath instance. The main user (who has highest priority) is routed from s to h with travel time 1 and then to t with bottleneck value 2 (solid (red) path). The whole illustration can be seen in Fig. 7.

Then none of the blocking users on the dashed (blue) paths can improve, since they receive their bottleneck value on their first arcs and the only possible change to arc (s, h) also gives a bottleneck value of at least 2, because the main user takes this arc with a higher priority at the same time. The main user has a bottleneck value of 2. We know from our last reduction to NarrowestPath that there is a path with bottleneck value 1 inside the network iff there is a Yes-instance for 3-SAT. So, exactly in this case there is a path the main user can change to and hence, the strategy profile is no Nash equilibrium. □

Furthermore, one can show analogously by reducing LongestPath to NashTest that the problem is also co-NP-hard in the single-commodity case in general graphs.

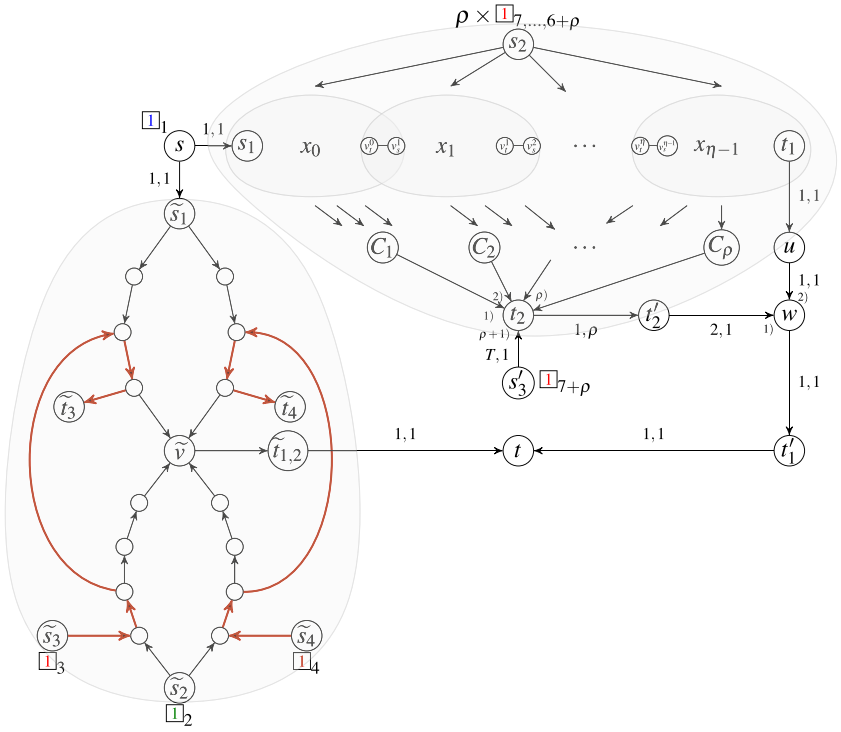


Fig. 8. Illustration of the reduction from 3-SAT to NashExistence. The first expression near an edge is its travel time, the second one its capacity. The small numbers at ingoing edges near vertices state the priorities of the edges. The upper graph is from the reduction from 3-SAT to NarrowestFlow used in the proof of Theorem 1, the left graph from Example 2. The exact travel times, capacities and priorities for the edges in these subgraphs can be found in the corresponding sections.

5.2. Complexity of the dynamic NashExistence problem

In this section we show that deciding whether a given instance has a Nash equilibrium or not is NP-hard for multi-commodity instances, even for unweighted users.

Theorem 5. *The* NashExistence *problem is* NP*-hard in the strong sense in the multi-commodity case, even if the graph is acyclic and all users are unweighted.*

Proof. The hardness is shown with a reduction similar to the one for the NarrowestFlow problem in the two-commodity case from Theorem 1. Again, an instance of 3-SAT is given by a set of Boolean variables $X = \{x_0, \ldots, x_{\eta-1}\}$, and ρ clauses $(C_j)_{j=1,\ldots,\rho}$ containing exactly three literals. The idea behind the subsequent construction follows from two former results:

We already know that the graph G_D constructed in the proof of Theorem 1 with one user routing for the first commodity can be traversed with a bottleneck value of 1 iff the instance of 3-SAT is a feasible truth assignment. Otherwise, the bottleneck is at least 2. On the other hand, without this user from the first commodity, the remaining $k - 1$ users can always route through the graph with bottleneck value 1 (independently on the instance of the 3-SAT problem).

Furthermore, we have seen in Example 2 an instance with a graph G_N that has no Nash equilibrium if there is a single user routing for its first commodity. Additionally, there is always a path with bottleneck value 1 for this user and never a path with smaller travel time. Without this user every feasible strategy of the other users gives a Nash equilibrium.

We add both graphs in one supergraph. First we add a super source s and connect it to the source s_1 of the first commodity in both graphs via an edge with travel time 1. Note that the vertices in the graph G_N are denoted with a tilde s.t. they can be distinguished from those in the graph G_D. We add a super terminal t and connect the terminal of the first commodity in G_N to this terminal via an edge with travel time 1. The capacities of all of these edges are set to 1. Next, we have to make sure that the user from the first commodity in G_D always suffers (i.e., has bottleneck value ≥ 2) if the flow has bottleneck value higher than 1. To that end we need to extend the instance a little bit, by adding vertices s'_3, t'_2, u, w and t'_1 and the following edges:

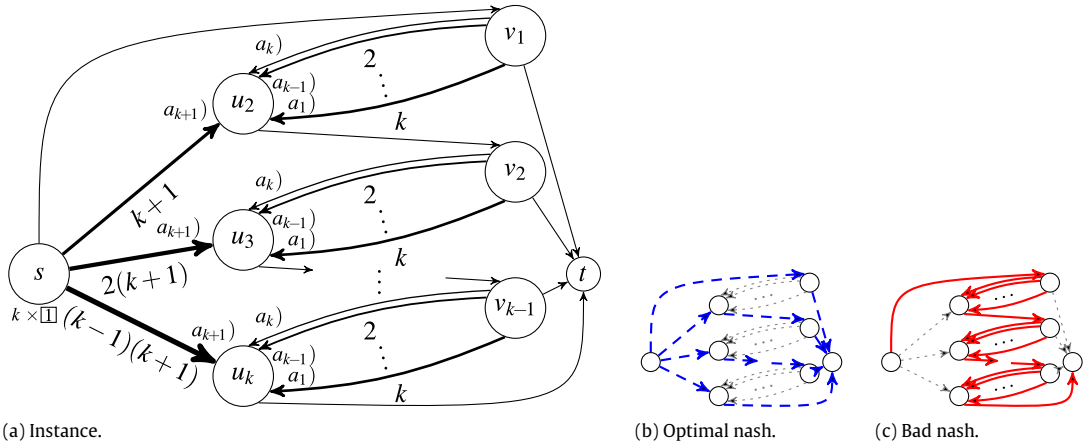
Edge	(s'_3, t_2)	(t_2, t'_2)	(t_2, w)	(t_1, u)	(u, w)	(w, t'_1)	(t'_1, t)
Travel time	T	1	2	1	1	1	1
Capacity	1	ρ	1	1	1	1	1

The former users from commodity 2 in G_D now route from s_2 to t'_2 and are called clause users. We add an additional blocking user, who routes from s'_3 to t'_1. The users from the first commodity in both graphs are merged to one user, who routes from the new super source s to the new super terminal t and is called the main user here. All other users from graph G_N have the same sources and terminals as before. The priorities are only important at a few vertices (indicated with small number and a closing bracket at the end of the edges in the figure): the edge (s'_3, t_2) receives the lowest priority of all edges entering t_2 and the edge (t_1, w) receives the lowest priority of all edges entering w. All other priorities are chosen arbitrarily. The whole illustration can be seen in Fig. 8.

Claim 1. We have seen in Example 2 that in the graph G_N there is always a path for the user from commodity 1 with bottleneck value 1. But then the user from the second commodity in this graph is not satisfied with her path.

Claim 2. The user from commodity 1 never routes along t_2, since the travel time on edge (t'_2, w) is 2 and, by Claim 1, there is always a path with travel time 1.

Claim 3. Iff the flow of the clause users in the subgraph G_D (with the modifications made above) has bottleneck value ≥ 2, then the user from commodity 1 cannot route through the subgraph with a bottleneck value equal to 1.



(a) Instance. (b) Optimal nash. (c) Bad nash.

Fig. 9. Instance with a bad Price of Anarchy (a). Every thick edge represents a path of a number of edges in a row, where the number is written near the edge. The small numbers with closing brackets near edges entering vertices denote the priorities of these edges. Two different strategy profiles are drawn on the right. Here, the dashed (blue) or solid (red) paths are used by the users, while the dotted (grey) paths are empty. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Proof. In the graph G_D with the modifications made above, the clause users arrive at vertex t_2 at time T if none of them is slowed down before. At this time the user from s'_3 also arrives at vertex t_2. Since her edge has a lower priority the user has to wait for one time unit. Then she arrives at vertex w at time $T+4$. On the other hand, if at least one of the clause users arrives at least one time unit later at t_2, then the user from s'_3 arrives at vertex w at time $T+3$.

The user from commodity 1 also traverses this vertex when routing through G_D. If she is not slowed down by any other user (in that case she has bottleneck value ≥ 2), then she arrives at vertex w at time $T+3$. Since her edge has a lower priority than the edge of the blocking user, she is slowed down by this user and receives bottleneck value at least 2, if this user also arrives there at time $T+3$. ◁

Main Claim. There is a feasible truth assignment for the 3-SAT instance iff there is Nash equilibrium in the enhanced instance.

Proof. Now assume that there is a feasible truth assignment for the 3-SAT instance. By the previous theorem we can route the user from commodity 1 along a path from s to t_1 with bottleneck value 1 and travel time T. The clause users can also be routed with bottleneck value 1 and travel time T to the vertex t_2. Since they have a better priority than the blocking user from s'_3, they reach their terminal t'_2 at time $T+1$, which is best possible. By Claim 3 the main user traverses the edge (w, t'_1) with travel time 1, since the blocking user is slowed down by the clause users. Hence, the main user arrives at her terminal t with bottleneck value equal to 1. Without the user from the first commodity in the graph G_N, there is also a Nash equilibrium in this graph. Hence, there is a Nash equilibrium for the enhanced instance.

Assume conversely that there is no feasible truth assignment for the 3-SAT instance. Then any flow with one user routing for the first commodity in the graph G_D has bottleneck value at least 2. By Claim 3 the main user receives bottleneck value ≥ 2, when routing through the subgraph G_D. Since there is always a path in G_N with bottleneck value 1 by Claim 1, she changes her path to this graph. Hence, there cannot be a Nash equilibrium in this case in the enhanced instance. □

5.3. Price of anarchy in games with bottleneck objective

In this section we give tight bounds on the Price of Anarchy and the strong Price of Anarchy. More precisely, we show that both are equal to the number of users both in the unweighted and in the weighted case. Note that besides the weight of the users, the free transit times, capacities and number of commodities are not important for our results.

For a set of instances $\mathcal{G}$ the *(strong) Price of Anarchy* is defined as the maximum ratio of the worst social objective of a (strong) Nash equilibrium in comparison to the social optimum, i.e., for a game with bottleneck objective we have

$$\mathsf{PoA}(\mathcal{G}) = \sup_{\Gamma\in\mathcal{G}} \sup_{f \text{ NE in } \Gamma} \frac{b(f)}{\mathrm{opt}(\Gamma)}$$

$$\mathsf{sPoA}(\mathcal{G}) = \sup_{\Gamma\in\mathcal{G}} \sup_{f \text{ SE in } \Gamma} \frac{b(f)}{\mathrm{opt}(\Gamma)}.$$

Theorem 6. *The (strong) Price of Anarchy is bounded from above by k.*

Proof. We show that $\mathsf{PoA} \leq k$, even in the weighted case. Let P be a path of a heaviest user with weight w in an optimal solution f^*. The bottleneck of the user is called $b \geq 1$ and the bottleneck of the strategy profile is $b^* \geq b$.

All edges in path P have at least capacity w (otherwise the user could not route there) and so every additional user cannot increase the latency by more than 1. If all k users take any of the edges of this path P at the same time, then the last one will receive a latency $\leq b+k-1$.

Assume that there is a Nash equilibrium f together with a user with bottleneck value higher than $b+k-1$. Then she can change to path P. Since at most all users can traverse the edges of that path at the same time, the bottleneck on that path is bounded from above by $b+k-1$. So f cannot be a Nash equilibrium and hence the ratio of the objective of a worst Nash equilibrium to the optimal objective is at most $(b+k-1)/b^* \leq 1+(k-1)/b^* \leq k$. □

Our next example shows that the above bound is in fact tight:

Example 4. Consider the graph from Fig. 9 with free transit times 1, capacities 1 and k unweighted users. There are two different strong equilibria.

The first strategy profile (see picture (b)) routes every user on one of the k disjoint paths denoted by $P_1 = (s, v_1, t)$, $P_j = (s, u_j, v_j, t)$ for $j \in \{2, \ldots, k-1\}$ and $P_k = (s, u_k, t)$. Then every user has a bottleneck value of 1, and hence the strategy profile is a strong equilibrium with bottleneck value 1.

We show that the strategy profile from picture (c) is also a strong equilibrium and has bottleneck value k. It routes user $j \in \{1, \ldots, k\}$ on path $\overline{P}_j = (s, v_1, u_2, v_2, \ldots, v_{k-1}, u_k, t)$. These paths

coincide on the edges (s, v_1), (u_j, v_j) and (u_k, t) and differ from each other only on the subpaths from v_{i-1} to u_i for $i \in \{2, \ldots, k\}$. We assume that path P_j always uses the subpath from v_{i-1} to u_i with priority a_j, i.e., the subpath consisting of $k - j + 1$ edges in a row.

So, user j with priority j is routed on path $\overline{P}_j$ with latency j on the first edge (s, v_1) and arrives in v_1 at time j. Afterwards she takes the subpath from v_1 to u_2 with $k-j+1$ edges and arrives in u_2 at time $k+1$ and so on. So, all users arrive at vertex u_i at time $(i-1)(k+1)$ for all $i \in \{2, \ldots, k\}$ and user j receives bottleneck value j for all $j \in \{1, \ldots, k\}$.

We show by induction on j that a coalition C of users who can change their strategies simultaneously in order to improve their bottleneck values cannot contain any of the users from 1 to j.

For $j = 1$ this is easy since user 1 has bottleneck value 1, and hence she has no incentive to change her strategy. The step is done from $j-1$ to j. We have to show that user j cannot be part of any improving coalition if users $q < j$ keep their strategies. Then user j has the highest priority of all users of the coalition. If user j wants to decrease her bottleneck value, she has to leave edge (s, v_1). Assume that user j changes to the edge (s, u_i) for some $i \in \{2, \ldots, k\}$. Note that all users $q < j$ are not affected by the change of the users of the coalition and arrive at vertex u_i at time $(i-1)(k+1)$ because of the structure of the graph. User j arrives there also at time $(i-1)(k+1)$, and her edge has a lower priority than those of users 1 to q. Since all of these users have to use edge (u_i, v_i) afterwards, user j receives at least bottleneck value j. So, she cannot improve by changing her path, and hence the strategy profile is a strong equilibrium with bottleneck value k. ◁

Note that this is exactly the same value as the Price of Anarchy in the static case (see e.g. [28]). But, the strong Price of Anarchy is much higher than in the static case, where it is given by 2. The reason behind this is the latency model. In the dynamic case simultaneous arrival worsens only the latter users in the equilibrium very badly, while the optimum solution can traverse the network with low cost. In the static case, however, users in a strong equilibrium suffer together and hence form a coalition to improve their situation.

6. Quickest flows

In this and the remaining sections we change the objective from the bottleneck to the sum case. We show that computing a flow with minimum makespan value can be done in polynomial time with the help of a maximum (dynamic) flow computation in the unweighted single-commodity case. Suppose that we are given a time horizon T. All flow has to start at time 0 in a source vertex s and reach the terminal vertex t by time T. The question of the maximum (dynamic) flow problem is how much flow can be sent from s to t in the specified time period. We stress that we look at an atomic variant of a continuous model, and it turns out that the flow sent at time θ is sent in the continuous model in the time interval $[\theta, \theta + 1)$.

We briefly repeat the necessary concepts of standard dynamic network flows. Let $G = (V, E)$ be a graph with a (single) source sink pair (s, t), free travel times τ_e and capacities u_e. For an edge $e \in E$, we denote by $\alpha(e)$ its starting vertex and by $\omega(e)$ its end vertex.

Let $f_e(\theta)$ be the amount of flow on edge e at time θ. Then the excess at vertex v for the flow f at time t is given by

$$\text{excess}_f(v, t) = \sum_{e \in \delta^-(v)} \sum_{\theta=0}^{t} f_e(\theta) - \sum_{e \in \delta^+(v)} \sum_{\theta=0}^{t} f_e(\theta), \tag{1}$$

where $\delta^+(v) \subseteq E$ are the edges emanating from v and $\delta^-(v) \subseteq E$ are those entering v.

Let the time horizon be T. Then $f_e(\theta) = 0$ for $\theta \notin [0, T - \tau_e]$.

Definition 4 (*Dynamic Network Flow*)**.** A dynamic network flow with time horizon T is a flow f with $0 \le f_e(\theta) \le u_e$ for all $e \in E$, $\text{excess}_f(v, t) \ge 0$ for all $v \in V \setminus \{s, t\}$, $t \in [0, T]$ and $\text{excess}_f(v, T) = 0$ for all $v \in V \setminus \{s, t\}$.

The amount of flow sent from the source to the sink is called value of the flow and it is given by $\text{val}(f) = -\text{excess}_f(s, T)$.

Definition 5. A *dynamic cut* with time horizon T is a function $X : [0, T + 1) \to 2^V$ s.t.

$$s \in X(\theta) \subseteq V \setminus \{t\} \quad \text{for all } \theta \in [0, T + 1)$$
$$X(\theta_1) \subseteq X(\theta_2) \quad \text{for } \theta_1 \le \theta_2.$$

For a dynamic cut X and a vertex $v \in V$ the earliest point in time when v enters X is given by $\xi_v := \min(\{\theta : v \in X(\theta)\} \cup \{T + 1\})$. Then $\xi_s = 0$ and $\xi_t = T + 1$. We define these values up to time $T + 1$, since the last units of flow have to enter the terminal t up to time T, and hence there is no more flow in the network at time $T + 1$.

A dynamic cut can be viewed as a cut which "grows over time". At any given time $\theta \in [0, T + 1)$ we have that $(X(\theta), V \setminus X(\theta))$ is a static s–t-cut. An edge $e = (u, v)$ *lies in* the dynamic cut if flow leaves vertex u at time ξ_u or later and enters vertex v before time ξ_v. The *capacity* U of a dynamic cut X with time horizon T equals the amount of flow that can be sent along the edges of G while they are in the cut, i.e.,

$$U(X) := \sum_{e=(u,v) \in E} \max\{0, \xi_v - \tau_e - \xi_u\} \cdot u_e. \tag{2}$$

It is well known that the maximum value of a dynamic flow and the minimum value of a dynamic cut coincide, see e.g. [9]. A central tool for finding a maximum dynamic flow is the concept of a temporally repeated flow:

Definition 6 (*Temporally Repeated Flow*)**.** Given a static flow $\hat{f}$ and a decomposition into a set of paths $\mathscr{P}$, which send $\hat{f}_P$ units of flow along path $P \in \mathscr{P}$. Then a *temporally repeated flow* f^T with time horizon T is defined as follows: for $P \in \mathscr{P}$ it sends during the time interval $[0, T - \tau_P]$ exactly $\hat{f}_P$ units of flow along path P, where $\tau_P = \sum_{e \in P} \tau_e$ is the length of path P. The total flow value sent by f^T is thus $\text{val}(f^T) = \sum_{P \in \mathscr{P}} \hat{f}_P (T - \tau_P)$.

It is also well-known that a maximum temporally repeated flow yields a maximum dynamic flow, see e.g. [32–34]. Thus, the value of a maximum temporally repeated flow equals the minimum value of a dynamic cut. To find a maximum temporally repeated flow one computes a minimum cost circulation $\hat{f}$ in the static model after introducing a back edge from t to s with infinite capacity and travel time $-(T+1)$ in polynomial time. Note that the cost of the back edge of $-(T + 1)$ is chosen because of the atomic model, while in the continuous case the cost is only $-T$.

Let us now proceed to flows in the deterministic queuing model. A dynamic flow f in the deterministic queuing model is given by a number of paths $(P_j)_{j=1,\ldots,k}$ and the number of users who are sent along these paths $(f_{P_j})_{j=1,\ldots,k}$.

We let $f_e^{\omega}(t) \ge 0$ be the amount of flow that leaves edge e to vertex $\omega(e)$ at time t, $f_e^{\alpha}(t) \ge 0$ be the amount of flow that enters edge e from the starting vertex $\alpha(e)$ at time t, $f_e(t) \ge 0$ be the amount of flow on edge e at time t, i.e.,

$$f_e(t) = \sum_{\theta=0}^{t} f_e^{\alpha}(\theta) - \sum_{\theta=0}^{t} f_e^{\omega}(\theta)$$

and $f_e^q(t)$ be the amount of flow that waits in the waiting queue of edge e at time t.

The flow entering an edge traverses it in time τ_e and enters the waiting queue, so we have:

$$f_e^q(t) = \max\left\{0,\, f_e^q(t-1) - u_e\right\} + f_e^\alpha(t-\tau_e). \tag{3}$$

The flow from the queue leaving an edge has to obey the capacity constraints, and thus

$$f_e^\omega(t) = \min\left\{u_e,\, f_e^q(t)\right\}. \tag{4}$$

We say that the dynamic flow f has *time horizon* T if all flow values $f_e(t), f_e^q(t), f_e^\alpha(t), f_e^\omega(t)$ are zero for $t > T$ and all edges $e \in E$. The *excess* in a vertex v at time t is defined as

$$\text{excess}_f(v,t) = \sum_{e\in\delta^-(v)}\sum_{\theta=0}^{t} f_e^\omega(\theta) - \sum_{e\in\delta^+(v)}\sum_{\theta=0}^{t} f_e^\alpha(\theta). \tag{5}$$

Since waiting in vertices is not allowed we have $\text{excess}_f(v,t) = 0$ for all $v \in V \setminus \{s,t\}$.

The *value* of a dynamic flow f in the deterministic queuing model is also defined to be $\text{val}(f) = -\text{excess}_f(s,T)$. Note that the model above actually reduces to the flow over time model if $f_e^\omega(t) = f_e^q(t) = f_e^\alpha(t-\tau_e)$. For the sake of an easier notation we set all flow and excess values to be zero for $t < 0$. It is easy to see that a maximum flow can be computed the same way as in the standard (atomic version of the) flow over time model by using temporally repeated flows. The only difference is that all flow starts at the source at time 0 and that flow beyond the capacity of an edge is able to wait in the waiting queue of an edge later on.

We now show that the capacity of any dynamic cut is an upper bound for the flow value of any flow in the dynamic queuing model. The proof is along the same lines as for the standard dynamic flow model and is contained in the Appendix for completeness. We note that the result cannot be deduced directly from the literature, since all flow has to leave the source s at time 0 in the deterministic queuing model and has to be stored in the some of the waiting queues later on.

Theorem 7. *For any dynamic flow f in the deterministic queuing model and any dynamic cut X (both with time horizon T) we have:* $\text{val}(f) \le U(X)$.

Proof. See Appendix A. □

The next step is to show that a *temporally repeated flow* in the *flow over time* model can be converted to a flow in the *deterministic queuing* model with the same flow value. Since there is a temporally repeated flow with the same flow value as the capacity of a dynamic cut the resulting flow is maximum. We stress here that, although we use the maximum dynamic flow from the standard model, the optimality result in our case does not follow directly from the literature, since the conversion has to take care about the queues and the buffering at the vertices.

For the conversion, we use the minimum cost circulation $\hat{f}$ which gives rise to the maximum temporally repeated flow in the standard dynamic flow model. Let $\mathscr{P}$ be the paths used by the circulation, $\hat{f}_P$ be the flow rate and τ_P the length of path $P \in \mathscr{P}$. Then f^T sends $\hat{f}_P$ units of flow at every time $\theta \in [0, T-\tau_P]$ along that path. The task of our algorithm is to send the same amount of flow along the same edges in such an order that they arrive within the time horizon in the deterministic queuing model. To that end we state the following definitions: $\mathscr{P}_e = \{P \in \mathscr{P} \mid e \in P\}$ are the paths traversing the edge $e \in E$, $F_P = \hat{f}_P \cdot (T + 1 - \tau_P)$ is the total flow sent along path P and $F = \sum_{P\in\mathscr{P}} F_P$ is the total amount of flow sent along all paths together, which equals the capacity of the minimal dynamic cut.

A general version of the algorithm can be seen in Algorithm 1 in Appendix A. The idea is the following. The algorithm allocates at least the same amount of flow as in the temporally repeated flow to all paths at the edges leaving the source. Then, this amount of flow is available for allocation at the subsequent edges. Here, it is allocated again as in the temporally repeated flow. To that end flow rates per time for points in time (from a list T_e) where some of the flow rates at edge e change are computed by the algorithm iteratively. Between two of these times from the list the flow rates remain constant. In the algorithm, first the flow rate is made available for the next edge e in path $P \in \mathscr{P}_e$ or the first edge of the path at the beginning. Then some amount of that rate is allocated to path P at edge e, which is then available for allocation at the subsequent edge e_P in path P. Here, a new entry stating the time and the amount of flow available is added to the list T_{e_P} of edge e_P.

Theorem 8. *Let f^T be a maximum temporally repeated flow with time horizon T. Then Algorithm 1 transforms f^T into a maximum flow in the deterministic queuing model in polynomial time.*

Proof. See Appendix A. □

6.1. The quickest flow problem

We now use the results about maximum dynamic flows in the deterministic queuing model to show that the problem of finding a flow with minimum makespan value is easy to solve in the unweighted single-commodity case.

Definition 7 (Quickest Flow *problem*)**.**

Input:	Instance $\Gamma = [G = (V,E),\ (\tau_e,\ u_e)_{e\in E},\ k,\ (s,\ t)]$ and a bound T.
Question:	Is there a feasible flow for that instance with makespan objective smaller or equal to T?

Theorem 9. *The* Quickest Flow *problem is solvable in polynomial time in the unweighted single-commodity case.*

Proof. Given an instance of the Quickest Flow problem, i.e., a graph, k units of flow, and a bound T for the travel time, the question is whether there is a feasible flow for k units of flow with makespan smaller or equal to T. By Theorem 8, a maximum flow with time horizon T can be computed in polynomial time. If the flow value is at least k, then we have found a solution for the Quickest Flow problem. If the flow value is smaller than k, then there is no solution for the Quickest Flow problem. □

Note that in the weighted case the problem becomes NP-complete with a reduction from the 3-Partition problem to the Quickest Flow problem in a network consisting of parallel links. In the multi-commodity case the problem also becomes NP-complete with a reduction from the 3-SAT problem as in [29] after introducing travel times in such a way that there is a feasible assignment iff there is a flow with makespan equal to some given value (see Appendix B.2).

7. Nash equilibria in games with sum objective

In this section we show that a Greedy type strategy computes a Nash equilibrium in a weighted single-commodity game with sum objective. For this reason, we restrict ourselves to the case that users with small weight have a better priority than users with larger weight (as in [20]). The reason is that users with a bad priority should not be able to use paths users with a good priority cannot use.

In an empty network there is always a shortest path with the property that every subpath is also a shortest path, since that case is equivalent to the static shortest path problem. Routing the first user with highest priority along such a path, a second user originating from the same source cannot enter a common

edge before the first user (here we need that the weight of the second user is not smaller than the weight of the first user, since otherwise she might overtake her on an edge the other user could not traverse). They might enter the edge at the same time, but because of the global priorities of the user, the first one leaves the common edge first and hence cannot be overtaken by the second user. Continuing with this approach gives an algorithm that iteratively routes users along a shortest path in the network, where the flow of the users with better priority is already routed. This is not an arbitrary shortest path, but one that arrives at every vertex in the path as early as possible. If there are multiple paths with the same travel time, one of them is picked arbitrarily. More formally:

Definition 8 (*Earliest Arrival Path*)**.** An *earliest arrival path* P with weight w in an empty network is an s–t-path s.t. for each vertex $v \in P$ the travel time of the subpath $P_{(s,v)}$ from s to v is minimum.

Let $f_{k-1} = (f_{Pj})_{j\in\{1,\dots,k-1\}}$ be a flow, where user $q \in \{1, \dots, k-1\}$ is routed on an earliest arrival s–t_q-path with weight w_q in the network where flow $f_{q-1} = (f_{Pj})_{j=1,\dots,q-1}$ is already routed.

Then an *earliest arrival path* P^k with weight w_k in the network with flow f_{k-1} already routed is an s–t_k-path s.t. for each vertex $v \in P^k$ the travel time of the subpath $P^k_{(s,v)}$ from s to v is minimum in the resulting flow $f = f_{k-1} + w_k\delta_{Pk}$. Here, δ_P denotes a flow that sends one unit of flow along path P and no other edge. Hence, $f = f_{k-1} + w_k\delta_{Pk}$ is the flow f_{k-1} with w_k additional units of flow on path P^k.

The resulting flow f is called an *earliest arrival path flow*.

The analysis above shows how to compute such an earliest arrival path flow. Hence, it always exists. By definition, a user with some priority on her path does not influence the travel times of all users with better priority, since she never overtakes any of them. That means on the other hand that the travel time of a user on an edge does not depend on the load of the other users with worse priority.

Observation 10. *Let $f_k = (f_{Pj})_{j\in\{1,\dots,k\}}$ be an earliest arrival path flow for k users. Then the travel time on each edge $e \in P^q$ for user q stays the same, if all users with inferior priority are removed, i.e., $\ell_{e,q}(f_q) = \ell_{e,q}(f_k)$.*

By the analysis above, a user who chooses to delay another user with lower priority on some edge does not have to suffer from the result of that choice later on, since the other user cannot overtake her. Hence, the travel time on the edges does not depend on the choice of previous edges. The result is stated more formally in the following:

Observation 11. *Let $f_k = (f_{Pj})_{j\in\{1,\dots,k\}}$ be an earliest arrival path flow for k users in the graph $G = (V, E)$, where P^j is the s–t_j-path of user $j \in \{1, \dots, k\}$ with weight w_j and $e = (v, w) \in P^k$ be an edge user k enters at time t. Furthermore, consider the graph $G' = (V, E \cup \{e', e''\})$ with $e' \notin E$ an additional edge from the source s to vertex v with travel time t and capacity 1 and $e'' \notin E$ an additional edge from vertex w to the terminal t_k with travel time 1 and capacity 1.*

Then the travel time of user k on edge e in the flow f in the graph G is equal to the travel time of user k on edge e in the flow f' in the graph G', where $f' = f - w_k\delta_{P^k} + w_k\delta_{\widetilde{P}^k}$ is the flow where user k changed from path P^k to the new path $\widetilde{P}^k = (s, e', v, e, w, e'', t)$.

From this second observation it follows immediately that an algorithm that aims at computing an earliest arrival path flow only needs the load of the users already traversing an edge, the load of the current user, and the time she enters the edge to compute her travel time on it. Her choices on previous edges and users added after her are irrelevant. Hence, an earliest arrival path can be computed with a slightly modified Dijkstra algorithm. By storing the load at the last point in time a user left an edge from the previous iterations, the current travel times can be computed easily. Note that the paths cannot be sorted by their length at the beginning, since the latency depends on the waiting time of the users who are already routed. So, it is not precisely a Greedy algorithm.

Proposition 1. *Every single-source game with sum-objective has a Nash equilibrium, even if the users are weighted.*

In particular, the earliest arrival path flow computed with an algorithm that iteratively computes earliest arrival paths gives such a Nash equilibrium in polynomial time.

Proof. With the observations above it follows immediately that the resulting flow is an earliest arrival path flow. So, we only have to show that an earliest arrival path flow is also a Nash flow. Take a look at some user j. By the observation above no user who was allocated after her can slow her down or any of the users who were allocated before her. So she and the users who were allocated before her have the same latency values they had when choosing their paths. Since she chooses a shortest path and no path length decreases due to additional users, she cannot change to another path and improve her objective. □

This shows that a Greedy type algorithm computes a Nash equilibrium in the weighted single source case. Note that this procedure does not work if there are users with large weight and good priority, since in that case they can be overtaken by users with small weight on an edge with small capacity. Furthermore, the procedure does not work in the weighted multi-commodity case, since an instance compared to Example 2 in the game with bottleneck objective can be constructed easily, which shows that also in the sum case there are instances without a Nash equilibrium (see Appendix B.1). Additionally, the NashTest problem is NP-hard to answer in the weighted single-commodity case or the unweighted multi-commodity case. These results can be shown with essentially the same reductions as in the bottleneck case. The NashExistence problem is also NP-hard to answer in the weighted multi-commodity case with essentially the same reduction as in the bottleneck case. We include the proof for NashExistence in Appendix B.3.

Comparing our results to Hoefer et al. [20] one sees that existence and computation complexity are very similar. However, the results cannot be obtained from their model for several reasons: in our model users only increase the travel time of users after them, while in [20] users entering an edge at the same time increase the travel time of all of these users. Moreover, the latency functions on edges in their model can be chosen in such a way that in any reasonable strategy profile an edge is used only by a single user, while an additional user in the deterministic queuing model increases the travel time only by a value of 1. Nevertheless, it is obvious that due to the atomic, dynamic setting in both models the computation complexity of a path with optimal objective in a network with other users already routed is NP-hard.

8. Conclusions

In this paper we investigated atomic dynamic routing games for a deterministic queuing model for sum and bottleneck objectives. We showed that quickest flows and Nash equilibria in the sum version can be computed efficiently, while the computation of narrowest paths/flows or Nash equilibria in the bottleneck version are NP-hard. Our work raises several challenging questions: first, what conditions ensure that Nash equilibria exist in the bottleneck case and when can such an equilibrium be computed efficiently? Second, how can the model be modified in such a way that in the atomic case more problems become tractable (and still the model can be considered "realistic")?

Appendix A. Maximum flows

Here we give the proofs from Section 6 that have been omitted before. The first lemma establishes some basic properties of dynamic flows in the deterministic queuing model.

Lemma 12. *For any dynamic flow f in the deterministic queuing model with time horizon T and any edge $e \in E$, and moment in time $t \geq 0$, it holds that the difference between flow entering edge e up to time $t - \tau_e$ and flow leaving the edge up to time t is given by the size of the waiting queue after time t, that is,*

$$\sum_{\theta=0}^{t-\tau_e} f_e^{\alpha}(\theta) - \sum_{\theta=0}^{t} f_e^{\omega}(\theta) = \max\left\{f_e^q(t) - u_e,\ 0\right\} \geq 0. \tag{A.1}$$

At the end (time horizon T) flow that entered edge e also left the edge, i.e.,

$$\sum_{\theta=0}^{T} f_e^{\alpha}(\theta) = \sum_{\theta=0}^{T} f_e^{\omega}(\theta). \tag{A.2}$$

The amount of flow leaving the outgoing edges from $v \in V$ after time $t + \tau_e$ must be at least as high as the amount of flow entering v after time t. More precisely

$$\sum_{e\in\delta^+(v)} \sum_{\theta=t+\tau_e}^{T} f_e^{\omega}(\theta) - \sum_{e\in\delta^-(v)} \sum_{\theta=t}^{T} f_e^{\omega}(\theta) \geq 0. \tag{A.3}$$

Proof. Let f be a dynamic flow in the deterministic queuing model with time horizon T, $e \in E$, $t \geq 0$. An easy calculation yields:

$$\begin{aligned}
\sum_{\theta=0}^{t-\tau_e} f_e^{\alpha}(\theta) &= \sum_{\theta=\tau_e}^{t} f_e^{\alpha}(\theta-\tau_e) = \sum_{\theta=0}^{t} f_e^{\alpha}(\theta-\tau_e)\\
&\overset{(3)}{=} \sum_{\theta=0}^{t} f_e^q(\theta) - \max\left\{0,\ f_e^q(\theta-1) - u_e\right\}\\
&= \sum_{\theta=0}^{t} f_e^q(\theta) - \sum_{\theta=0}^{t} \max\left\{0,\ f_e^q(\theta-1) - u_e\right\}\\
&= \sum_{\theta=0}^{t} f_e^q(\theta) - \sum_{\theta=0}^{t-1} \max\left\{0,\ f_e^q(\theta) - u_e\right\}\\
&= \sum_{\theta=0}^{t} f_e^q(\theta) - \sum_{\theta=0}^{t} \max\left\{0,\ f_e^q(\theta) - u_e\right\}\\
&\quad + \max\left\{f_e^q(t) - u_e,\ 0\right\}.
\end{aligned} \tag{A.4}$$

If $f_e^q(\theta) > u_e$, then we have:

$$\begin{aligned}
f_e^q(\theta) - \max\{0, f_e^q(\theta) - u_e\} &= f_e^q(\theta) - (f_e^q(\theta) - u_e) = u_e\\
&= \min\{u_e, f_e^q(t)\} \overset{(4)}{=} f_e^{\omega}(\theta).
\end{aligned}$$

On the other hand, if $f_e^q(\theta) < u_e$, then

$$\begin{aligned}
f_e^q(\theta) - \max\{0, f_e^q(\theta) - u_e\} &= f_e^q(\theta) = \min\{f_e^q(\theta), u_e\}\\
&\overset{(4)}{=} f_e^{\omega}(\theta).
\end{aligned}$$

Using the above two calculations in (A.4) now yields the first claim of the lemma.

We proceed with the second claim. After the last time step the waiting queue has to be empty. So we must have that $f_e^q(T) \leq u_e$. Furthermore, any flow entering after time $T - \tau_e$ cannot reach the terminal vertex up to time T. So $f_e^{\alpha}(\theta) = 0$ for $\theta > T - \tau_e$. Using these two properties in (A.1) implies the second claim.

For the last claim, observe that for any vertex $v \in V \setminus \{s, t\}$:

$$\begin{aligned}
0 &= \operatorname{excess}_f(v, t)\\
&\overset{(5)}{=} \sum_{e\in\delta^-(v)} \sum_{\theta=0}^{t} f_e^{\omega}(\theta) - \sum_{e\in\delta^+(v)} \sum_{\theta=0}^{t} f_e^{\alpha}(\theta)\\
&\overset{(A.1)}{\leq} \sum_{e\in\delta^-(v)} \sum_{\theta=0}^{t} f_e^{\omega}(\theta) - \sum_{e\in\delta^+(v)} \sum_{\theta=0}^{t+\tau_e} f_e^{\omega}(\theta). \qquad (*)
\end{aligned}$$

So, for all $0 \leq t \leq T$ this implies that

$$\begin{aligned}
0 &= \operatorname{excess}_f(v, T)\\
&\overset{(5)}{=} \sum_{e\in\delta^-(v)} \sum_{\theta=0}^{T} f_e^{\omega}(\theta) - \sum_{e\in\delta^+(v)} \sum_{\theta=0}^{T} f_e^{\alpha}(\theta)\\
&\overset{(A.2)}{=} \sum_{e\in\delta^-(v)} \sum_{\theta=0}^{T} f_e^{\omega}(\theta) - \sum_{e\in\delta^+(v)} \sum_{\theta=0}^{T} f_e^{\omega}(\theta)\\
&= \underbrace{\sum_{e\in\delta^-(v)} \sum_{\theta=0}^{t-1} f_e^{\omega}(\theta) - \sum_{e\in\delta^+(v)} \sum_{\theta=0}^{t+\tau_e-1} f_e^{\omega}(\theta)}_{\geq 0 \text{ by } (*)}\\
&\quad + \sum_{e\in\delta^-(v)} \sum_{\theta=t}^{T} f_e^{\omega}(\theta) - \sum_{e\in\delta^+(v)} \sum_{\theta=t+\tau_e}^{T} f_e^{\omega}(\theta)\\
&\geq \sum_{e\in\delta^-(v)} \sum_{\theta=t}^{T} f_e^{\omega}(\theta) - \sum_{e\in\delta^+(v)} \sum_{\theta=t+\tau_e}^{T} f_e^{\omega}(\theta).
\end{aligned}$$

This settles the claim. □

With the lemma from above we show that the capacity of any dynamic cut is an upper bound for the flow value of any flow in the dynamic queuing model as stated in Theorem 7.

Theorem 13. *For any dynamic flow f in the deterministic queuing model and any dynamic cut X (both with time horizon T) we have:* $\operatorname{val}(f) \leq U(X)$.

Proof. First, note that

$$\sum_{e\in\delta^+(t)} \sum_{\theta=\xi_t+\tau_e}^{T} f_e^{\omega}(\theta) - \sum_{e\in\delta^-(t)} \sum_{\theta=\xi_t}^{T} f_e^{\omega}(\theta) = 0, \tag{A.5}$$

since $\xi_t = T + 1$ and hence there are in fact no summands left.

For any dynamic flow f in the deterministic queuing model and any cut X with times ξ_v for $v \in V$, where $\xi_s = 0$, we have:

$$\begin{aligned}
0 \leq \operatorname{val}(f) &= -\operatorname{excess}_f(s, T)\\
&= \sum_{e\in\delta^+(s)} \sum_{\theta=0}^{T} f_e^{\omega}(\theta) - \sum_{e\in\delta^-(s)} \sum_{\theta=0}^{T} f_e^{\omega}(\theta).
\end{aligned}$$

We use $f_e^{\omega}(\theta) = 0$ for $e \in \delta^+(s)$ and $\theta < \tau_e$, add Eq. (A.3) for all $v \in V \setminus \{s, t\}$ together with the equation from above for vertex t:

$$\begin{aligned}
&\leq \sum_{v\in V} \left(\sum_{e\in\delta^+(v)} \sum_{\theta=\xi_v+\tau_e}^{T} f_e^{\omega}(\theta) - \sum_{e\in\delta^-(v)} \sum_{\theta=\xi_v}^{T} f_e^{\omega}(\theta) \right)\\
&\leq \sum_{\substack{e\in E\\ e=(u,v)}} \sum_{\theta=\xi_u+\tau_e}^{T} f_e^{\omega}(\theta) - \sum_{\substack{e\in E\\ e=(u,v)}} \sum_{\theta=\xi_v}^{T} f_e^{\omega}(\theta)\\
&\leq \sum_{\substack{e\in E\\ e=(u,v)}} \sum_{\theta=\xi_u+\tau_e}^{\xi_v-1} f_e^{\omega}(\theta)
\end{aligned}$$

$$\leq \sum_{\substack{e\in E\\ e=(u,v)}} \max\{\xi_v - \xi_u - \tau_e,\ 0\}\, u_e = U(X). \quad \square$$

The next goal is to show that a *temporally repeated flow* in the *flow over time* model can be converted to a flow in the *deterministic queuing* model with the same flow value. A general version of the algorithm can be seen in Algorithm 1.

Algorithm 1: Flow computation. More details for * in Remark 1.

Input: Graph $G = (V, E)$, flow rates $\hat{f}_p$ and path length τ_P of paths $P \in \mathscr{P}$ in a temporally repeated flow and time horizon T.

Output: Flow in deterministic queuing model.

1 For $e = (v, w) \in E$ set $T_e = \emptyset$ and add entry $\left(T + 1 - \tau_{P_{(v,t)}};\ (0_P)_{P\in\mathscr{P}_e}\right)$ to T_e and for $e \in \delta^+(s)$ add $\left(0;\ (0_P)_{P\in\mathscr{P}_e}\right)$ to T_e;

2 **for** *edge* $e = (v, w) \in E$ *(sorted *1)* **do**

3 Set the active paths $\mathscr{P}'_e := \mathscr{P}_e$;

4 **if** $v = s$ **then** for all $P \in \mathscr{P}'_e$ set the amount in the buffer $W_P = F_P$ **else** for all $P \in \mathscr{P}'_e$ set $W_P = 0$ **while** $\mathscr{P}'_e \neq \emptyset$ **do**

5 Remove the entry $\left(t_\alpha;\ (X_P)_{P\in\mathscr{P}_e}\right)$ from T_e, where t_α is the smallest time in T_e. Let t_ω be the second smallest time in T_e and $\theta = t_\omega - t_\alpha$ be the length of the current time interval.

6 For $P \in \mathscr{P}'_e$ let $f_P = \min\left\{\hat{f}_P,\ X_P + W_P\right\}$ be the flow value of path $P \in \mathscr{P}'_e$ that can be utilized at edge e and $u = \min\left\{u_e,\ \sum_{P\in\mathscr{P}'_e}(X_P + W_P)\right\}$ be the capacity of edge e that can be utilized;

7 **if** $\theta > \widetilde{\theta}$, *where* $\widetilde{\theta}$ *is the maximum time span the flow value* f_P *and the capacity* u *can be utilized (*2)* **then** add $\left(t_\alpha + \tilde{\theta};\ (X_P)_{P\in\mathscr{P}_e}\right)$ to T_e and set $\theta := \widetilde{\theta}$ find $(Y_P)_{P\in\mathscr{P}'_e}$ with $f_P \leq Y_P \leq X_P + \left\lfloor \frac{W_P}{\theta} \right\rfloor$, $Y_P \in \mathbb{N}$ for all $P \in \mathscr{P}'_e$ and $\sum_{P\in\mathscr{P}'_e} Y_P = u$;

8 for all $P \in \mathscr{P}'_e$ add $\left(t_\alpha + \tau_e;\ (Y_P)_{P\in\mathscr{P}'_e}\right)$ to T_{e_P}, where e_P is the subsequent edge after e in P;

9 **for** $P \in \mathscr{P}'_e$ **do**

10 **if** F_P *units of flow have been allocated to path* P *at edge* e **then** remove path P from $\mathscr{P}'_e$ and add $(t_\omega + \tau_e;\ 0_P)$ to the list T_{e_P}

11 for all $P \in \mathscr{P}'_e$ the remaining units of flow available for path P, i.e., $\theta\, X_P + W_P - Y_P$, are stored as the new buffer W_P and are available for allocation in the upcoming iteration;

Some explanations for the algorithm are given in the following remark:

Remark 1.
- T_e: The algorithm computes inflow rates X_P for all paths $P \in \mathscr{P}_e$ for every edge e. For better reading, rates from previous iterations are denoted by X, rates that are currently computed and stored for later iterations are denoted by Y. For all times t where one of the entries from $(X_P)_{P\in\mathscr{P}_e}$ changes or a path becomes inactive at e we store an entry $(t;\ (X_P)_{P\in\mathscr{P}_e})$ in a list T_e sorted by the time in an increasing order. So, starting at time t all paths $P \in \mathscr{P}_e$ have flow values X_P that state how many units of flow are available to be sent along edge e per time unit until the next time in the list T_e. For simplification of notation we define the following: if for time t flow values for some paths $P \in \mathscr{P}_e$ are missing, i.e., we have an entry $(t, (X_P)_{P\in\mathscr{P}'})$ for some strict subset $\mathscr{P}' \subset \mathscr{P}_e$, then they are set to the previous flow value for P in T_e, i.e., the flow value for P for the maximum time $t' \leq t$ for which there is a flow value for path P or to 0 if there is no such value. So we can assume that all entries we remove of T_e have the form $(t, (X_P)_{P\in\mathscr{P}_e})$, even if we only added $(t, (X_P)_{P\in\mathscr{P}'})$ for some subset $\mathscr{P}' \subseteq \mathscr{P}_e$ to T_e before. During the algorithm we add entries (t, Y_P) to T_e. This is done under the constraint that an entry with the same flow value has not been added before. That means we only add the entry, if the flow value for P for the maximum time $t' \leq t$ for which there is a flow value for path P differs from the current value Y_P. Furthermore, the list never contains a time twice.
  Note that flow starting at time $T + 1 - \tau_{P_{(v,t)}}$ added to T_e for $e = (v, w) \in E$ in Step 1 cannot reach the terminal vertex t up to the time horizon T. We show in Lemma 15 that the total amount of flow entering an edge e in path P in the temporally repeated flow up to time t is also allocated to P at e up to time t. Hence, there is no unit of flow available for allocation at edge e at time $T + 1 - \tau_{P_{(v,t)}}$.
- *1 in Step 2: The graph $G' = (V, E')$ that consists of the edges from the paths $\mathscr{P}$ of the path decomposition of a maximum temporally repeated flow $\hat{f}$ is acyclic. This follows from the fact that the maximum temporally repeated flow is obtained via a flow decomposition of a minimum cost flow (see e.g., [34]). Since we assumed that all travel times are strictly positive, a cycle in the decomposition would contradict the optimality of the minimum cost flow. Thus, we can assume that the vertices in G' are numbered according to a topological sorting, and that in Step 2 the edges are considered in the order of the topological sorting of their starting vertices.
- *2 in Step 7: The maximum time span $\widetilde{\theta}_P$ the flow value f_P of path P can be utilized can be computed as follows: if the amount of flow available per time X_P is at least as high as the flow value f_P, i.e., $X_P \geq f_P$, then this value can be utilized for the whole time interval, i.e., $\theta_P = \theta$. Otherwise, $X_P < f_P$. Then the length of the time interval is decreased to some value θ_P. The minimum amount of flow $(f_P\, \theta_P)$ allocated during the interval (of length θ_P) cannot exceed the amount of flow available $(W_P + \theta_P X_P)$ during that interval. This yields $\theta_P = \left\lfloor \frac{W_P}{f_P - X_P} \right\rfloor$ in that case. Furthermore, we know by step 6 that $f_P \leq X_P + W_P$. With $X_P < f_P$ we get $\widetilde{\theta}_P = \frac{W_P}{f_P - X_P} \geq 1$.
- The maximum time span $\widetilde{\theta}_e$ the capacity u can be utilized can be computed as follows: if the amount of flow available per time $X_e = \sum_{P\in\mathscr{P}'_e} X_P$ is at least as high as the capacity u, i.e., $X_e \geq u$, then this value can be utilized for the whole time interval, i.e., $\theta_e = \theta$. Otherwise, $X_e < u$. Then the length of the time interval is decreased to some value θ_e. The minimum amount of flow $(u\, \theta_e)$ allocated during the interval (of length θ_e) cannot exceed the amount of flow $(\sum_{P\in\mathscr{P}'_e}(W_P + \theta_e X_P))$ available during that interval. This yields $\theta_e = \left\lfloor \frac{\sum_{P\in\mathscr{P}'_e} W_P}{u - X_e} \right\rfloor$ in that case. Furthermore, we know by step 6 that $u \leq \sum_{P\in\mathscr{P}'_e}(X_P + W_P)$. So $\widetilde{\theta}_e \geq 1$.
  Then $\widetilde{\theta} = \min\left\{\theta_e, \theta_P \mid P \in \mathscr{P}'_e\right\}$.
- For the last edge e in a path P there is no subsequent edge e_P. So, the amount is stored for the terminal vertex t instead.

Lemma 14. *The integer program in Step* 7 *of the algorithm has a solution, which can be computed in time* $\mathcal{O}(m)$.

Proof. For the ease of notation we write $b_P = X_P + \left\lfloor \frac{W_P}{\theta} \right\rfloor$. First we show that the program is feasible, i.e., $\sum_{P\in\mathscr{P}'_e} f_P \leq u \leq \sum_{P\in\mathscr{P}'_e} b_P$. The second inequality follows directly from the definition of $u \leq$

$\sum_{P\in\mathscr{P}'_e} b_P$. To see the first inequality we distinguish between two cases. Let $u = u_e$. By definition we have $f_P \le \hat{f}_P$. A temporally repeated flow meets the capacity constraints of the network, i.e., $\sum_{P\in\mathscr{P}_e} \hat{f}_P \le u_e$. So, we get $\sum_{P\in\mathscr{P}_e} f_P \le u_e = u$. Now, let $u = \sum_{P\in\mathscr{P}'_e} b_P$. To see the first inequality we partition $\mathscr{P}'_e = \mathscr{P}_1 \cup \mathscr{P}_2$ s.t. $f_P = \hat{f}_P \le b_P$ for $P \in \mathscr{P}_1$ and $f_P = b_P$ for $P \in \mathscr{P}_2$. Then $\sum_{P\in\mathscr{P}'_e} f_P = \sum_{P\in\mathscr{P}_1} \hat{f}_P + \sum_{P\in\mathscr{P}_2} b_P \le \sum_{P\in\mathscr{P}'_e} b_P = u$.

The integer program is feasible and can be written in the following form: $\sum_{P\in\mathscr{P}'_e} Y_P = u, f_P \le Y_P \le b_P$ for all $P \in \mathscr{P}'_e, Y_P \in \mathbb{N}$, where all data is integral. It is easy to see that the corresponding matrix is totally unimodular (see e.g., [35]). Hence, a solution can be computed efficiently. Furthermore, a solution can also be given explicitly by assigning the minimum value to the all paths and distributing the remaining amount of flow arbitrarily among the paths without violating the upper bounds. In fact, this can be done in linear time $\mathcal{O}(m)$. □

The following lemma states that the algorithm transforms a maximum temporally repeated flow to a flow in the deterministic queuing model which sends the same amount of flow within the same time horizon.

Lemma 15. *The dynamic flow in the deterministic queuing model with time horizon T constructed by Algorithm 1 has the same flow value as the maximum temporally repeated flow with time horizon T used as input for the algorithm.*

Proof. First we show that the algorithm is well-defined in the sense that every step can be carried out. We show some properties by induction on the number of edges in a path P considered so far in the for-loop of Step 2. To that end let $\widetilde{T}_e = \{t_1, \dots, t_\mu\}$ be the set of all times added to the list T_e for edge $e = (v, w)$ during the algorithm with $t_i < t_{i+1}$ for $i \in \{1, \dots, \mu - 1\}$. We show by induction on t with $t + 1 \in \widetilde{T}_e$ the following statements:

(1) The algorithm allocates at least $d_{P,e,t} = \hat{f}_P \cdot \min\{t + 1 - \tau_{P_{(s,v)}}, T+1-\tau_P\}$ units of flow to path P at edge $e = (v, w) \in P$ up to time t, which is the amount the temporally repeated flow f^T with time horizon T routes along path P at edge e. In particular, it allocates $F_P = \hat{f}_P(T + 1 - \tau_P)$ units of flow to P at edge e up to time $T - \tau_{P_{(v,t)}}$.
(2) At a start of the while-loop, the list T_e in Step 6 of the algorithm consists of at least one time $t_\alpha < T + 1 - \tau_{P_{(v,t)}}$ where the flow rate X_P or the amount of flow in the buffer W_P is greater than zero. Furthermore, it consists of time $(T + 1 - \tau_{P_{(v,t)}})$.
(3) The lower bound computed in Step 6 fulfils $f_P > 0$ and the solution of the integer program in Step 7 fulfils $Y_P > 0$.
(4) $\tau_{P_{(s,v)}}$ is the smallest time in $\widetilde{T}_e$ and there is some amount of flow available for allocation at that time, i.e., $X_P + W_P > 0$ for these variables at that time.

Induction start (for the edges): We start with the first edge $e = (s, u)$ of path P. Then $\widetilde{T}_e$ includes times $\tau_{P_{(s,s)}} = 0$ and $T+1-\tau_P > 0$. Since there is never a smaller time added to T_e, 0 is the smallest time in $\widetilde{T}_e$. The induction on the time starts with the first time t with $t + 1 \in \widetilde{T}_e$, i.e., $t = t_1 - 1 = -1$. At that time no flow is sent in both models and hence there is also no iteration and nothing to show. At the start of the iteration for edge $e = (s, u)$ (at time t_1) the buffer W_P is set to the total amount of flow F_P that has to be allocated to P at e. The amount in the buffer is reduced in Step 11 by the amount allocated to path P.

For the induction step (for the time) at the first edges consider some time $t = t_{i+1} - 1$ with $t_{i+1} \in \widetilde{T}_e, i \in \{1, \dots, \mu - 1\}$ and the iteration from time t_i up to time $t = t_{i+1} - 1$. Let $D \ge 0$ be the amount of flow that has already been allocated to P at e up to time $t_i - 1$. If all F_P units of flow have already been allocated to path P up to time $t_i - 1$, then we are done, since statement (1) is satisfied and no more iteration of the while-loop is executed in the algorithm. Otherwise, the if-condition in Step 10 was not satisfied in the previous iteration that ended at time $t_i - 1$. So, we consider time t_i in T_e at the start of the while-loop. The amount of flow in the buffer at that time is positive, i.e., $W_P > 0$. Since $t = t_{i+1} - 1$, there is also time $T + 1 - \tau_P = t_\mu \ge t_{i+1} = t + 1 > t$ in the set T_e, which shows (2). We have to show that at least $d_{P,e,t_{i+1}-1} = \hat{f}_P\, t_{i+1}$ units of flow are allocated to P at e up to time $t_{i+1} - 1$. By induction hypothesis on t we know that $D \ge d_{P,e,t_i-1} = \hat{f}_P\, t_i$ units of flow have already been allocated to P at e up to time $t_i - 1$. If $W_P \ge \hat{f}_P$, then also $Y_P \ge f_P = \hat{f}_P > 0$ (showing (3)) and hence, at least f_P units of flow are allocated to P at e per time. So, in total at least $d_{P,e,t_i-1} + \hat{f}_P(t_{i+1} - 1 + 1 - t_i) = \hat{f}_P\, t_{i+1} = d_{P,e,t_{i+1}-1}$ units of flow are allocated to P at e up to time $t_{i+1} - 1$. Otherwise $W_P < \hat{f}_P$. Then $D > F_P - \hat{f}_P$ and less than $\hat{f}_P$ units of flow remain to be allocated. Then $Y_P = f_P = W_P > 0$ (showing (3)) and hence, at least W_P units of flow are allocated to P at e per time. So, in total at least $F_P \ge d_{P,e,t_{i+1}-1}$ units of flow are allocated to P at e up to time $t_{i+1} - 1$. In particular, F_P units of flow are allocated to P at e up to time $T + 1 - \tau_P$ and hence, no larger time is added to $\widetilde{T}_e$.

Induction step (for the edges): Consider path P at some edge $e = (v, w) \in E \setminus \delta^+(s)$. By induction hypothesis on e, for the previous edge $e' = (u, v)$ in P before e we have $t'_1 = \tau_{P_{(s,u)}}$ as the smallest and $t'_{\mu'} = T + 1 - \tau_{P_{(u,t)}}$ as the largest time in the set $\widetilde{T}_{e'}$ by (4). Furthermore, the first $Y_P > 0$ units of flow are made available for allocation for P at e at time $\tau_{P_{(s,v)}}$. Hence, for the induction start at a time smaller than $\tau_{P_{(s,v)}}$ there is nothing to show.

For the induction step (for the time) at the edge e consider some point in time $t = t_{i+1} - 1$ with $t_{i+1} \in \widetilde{T}_e, i \in \{1, \dots, \mu - 1\}$ and the iteration from time t_i up to time $t = t_{i+1} - 1$. Let $D \ge 0$ be the amount of flow that has already been allocated to P at e up to time $t_i - 1$ and consider time t_i in T_e at the start of the while-loop. If all F_P units of flow have already been allocated to path P up to time $t_i - 1$, then we are done, since statement (1) is satisfied and no more iteration of the while-loop is executed in the algorithm. Otherwise, the if-condition in Step 10 was not satisfied in the previous iteration that ended at time $t_i - 1$ (or the current iteration is the first one).

If the flow rate satisfies $X_P > 0$ during the iteration starting at time t_i, then T_e includes time t_i with $X_P > 0$ as claimed. Otherwise $X_P = 0$. Then, the last iteration for path P at the previous edge e' ended before time $t_i - \tau_{e'}$, since otherwise $Y'_P > 0$ units of flow would have been made available for time t_i in that iteration (by (3)). So, by induction hypothesis, F_P units of flow have been allocated to P at e up to time $t_i - 1 - \tau_{e'}$. Hence, they are available for allocation at e up to time $t_i - 1$. Since they have not been allocated to P at e so far, the remaining amount is stored in the buffer, i.e., $W_P > 0$. So, we always have $X_P + W_P > 0$ at the start of the while-loop. With $\hat{f}_P > 0$ this implies that $f_P > 0$. Since the integer program always has a feasible solution we have $Y_P \ge f_P > 0$. This shows (3).

By induction hypothesis on the time t we know that $D \ge d_{P,e,t_i-1}$ units of flow have already been allocated to P at e up to time $t_i - 1$. By induction hypothesis on e we know that at least $d_{P,e',t-\tau_{e'}} = d_{P,e,t}$ units of flow have been allocated to the previous edge e' in P before e up to time $t - \tau_{e'}$ and hence are available for allocation to P at e up to time t. If $D \ge d_{P,e,t}$, then we are done.

Otherwise, $D < d_{P,e,t}$. But we know by the argumentation above that the amount of flow available for allocation to P at e up to time t is at least $d_{P,e,t}$. Hence, at least the missing $d_{P,e,t} - D$ units of flow are available for allocation up to time t, i.e., in the algorithm we have $\theta X_P + W_P \ge d_{P,e,t} - D$ with $\theta = t_{i+1} - t_i = t + 1 - t_i$. If $X_P + W_P \ge \hat{f}_P$, we are also done, because then at least $f_P = \hat{f}_P$ units of flow are allocated by the algorithm per time between t_i and t, i.e., in total at

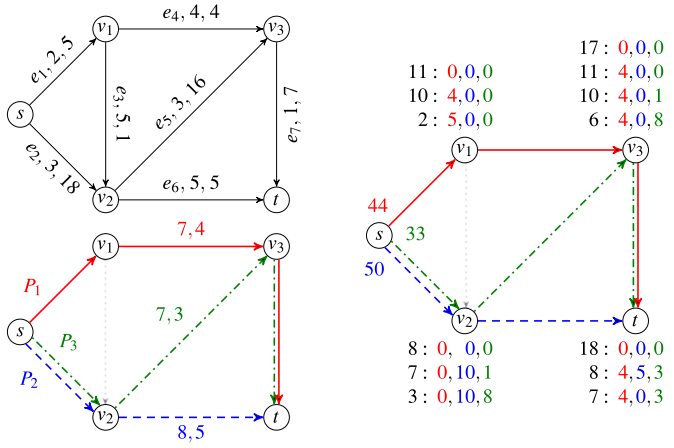


Fig. A.10. Example for the calculation of a maximum flow. Graph with edge names, travel times and capacities on the top left, path decomposition from the *flow over time* model (with length and capacity) on the bottom left and functioning of Algorithm 1 on the right. The numbers near the paths give the numbers of users. At each vertex v and every point in time the flow rates change an entry $(t : Y_{P_1}, Y_{P_2}, Y_{P_3})$ is drawn. It states the time t the flow reaches the vertex (and the outgoing edges) and the amount Y_{P_j} allocated to the corresponding path P_j at the ingoing edge per time unit until the next time entry for all $j \in \{1, 2, 3\}$, which is then available for allocation at the outgoing edge. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

least $d_{P,e,t_i-1} + \hat{f}_P\,\theta = \hat{f}_P(t_i - \tau_{P_{(s,v)}} + \theta) = \hat{f}_P(t + 1 - \tau_{P_{(s,v)}}) = d_{P,e,t}$ units of flow up to time t.

Else, we have $f_P = X_P + W_P < \hat{f}_P$. Then all units of flow that have been made available for P at e up to time t_i (at least d_{P,e,t_i} units of flow), except for those f_P units above, have already been allocated to P. If $t_i = t$, then $D + X_P + W_P$ is exactly the amount of flow available for allocation for P at e up to time t. We already know that (by induction hypothesis on e) it is at least $d_{P,e,t}$. Since it is allocated in the current iteration we are done.

Otherwise, we have $t_i < t$, i.e., $t_i < t < t_{i+1}$. Since there is no time in the list $\widetilde{T}_e$ in between and $Y_P \geq f_P = X_P + W_P$, the buffer is empty at time $t_i + 1 \leq t$. Since the inflow rate X_P does not change (otherwise there would be another time in $\widetilde{T}_e$) we have $W_P = 0$ at time t_i. Hence, $f_P = X_P$ and $W_P = 0$ during the whole interval. So, $D + (t + 1 - t_i)\,X_P$ is exactly the amount of flow available for allocation for P at e up to time t. We already know that (by induction hypothesis on e) it is at least $d_{P,e,t}$ and no unit of flow is stored in the buffer. Hence, $d_{P,e,t}$ units of flow are allocated to path P at edge e up to time t by the algorithm. □

Next we give an example of the algorithm.

Example 5. Consider the instance from Fig. A.10 for the maximum s–t-flow problem with time horizon $T = 17$ in the deterministic queuing model.

In the *flow over time* model, a maximum temporally repeated flow routes for exactly $T + 1 - \tau_{P^1} = 11$ time units $\hat{f}_{P^1} = 4$ units of flow along path $P^1 = (s, v_1, v_3, t)$ (solid (red)), i.e., in total $F_{P^1} = 11 \cdot 4 = 44$ units, for $T + 1 - \tau_{P^2} = 10$ time units $\hat{f}_{P^2} = 5$ units of flow along path $P^2 = (s, v_2, v_3, t)$ (dashed (blue)), i.e., in total $F_{P^2} = 10 \cdot 5 = 50$ units, and for exactly $T + 1 - \tau_{P^3} = 11$ time units $\hat{f}_{P^3} = 3$ units of flow along path $P^3 = (s, v_2, t)$ (dash-dotted (green)), i.e., in total $F_{P^3} = 11 \cdot 3 = 33$ units, which gives a flow with flow value of $\mathrm{val}(f^{T+1}) = 127$.

The algorithm works as stated in the following table. It states the following information: the active path denoted by P^j for some j, the amount of flow X_{Pj} available per time unit during the current interval, the lower bound f_P for the flow value of path P^j and the for the capacity u of the edge computed by the algorithm, the initial length θ and the real length $\widetilde{\theta}$ of the current time interval. If the length of the interval is reduced, a new point in time is added to the current list T_{e_i}. There is an entry Y_{Pj} for the solution of the integer program, the subsequent edge e_P in P after e and the entry added to the list $T_{e_{pj}}$ of that edge. The last entries state the amount of flow W_{Pj} in the buffer at the end of the iteration and the amount of flow $\widetilde{F}_{Pj}$ that still has to be allocated to P at e.

The algorithm iterates over all edges (every iteration is a block in the table). The first line of every block states the current edge e_i and the starting conditions for the current iteration. For the source the paths P^j traversing the edge have some flow in the butter W_{pj} at the beginning, otherwise the buffer is empty. Then the important data for every iteration over the time is stated explicitly in the middle lines of each block. If there is more than one active path, then for one time there are multiple lines, one for each path. The most important column is the 10th (add to $T_{e_{pj}}$), where Step 8 of the algorithm adds new times and flow values to the list of the upcoming edge e_{pj} of path P^j. In the last line of each block the iteration ends because $\widetilde{F}_{pj} = 0$ and additional times at which no more flow is sent are added to the list. See Table A.1.

Some entries in the table for the list $T_{e_{pj}}$ seem to be missing. That is because the algorithm only adds entries in step 8 if they differ from the previous entry in the list. Since the flow rates do not change, no new time step is necessary.

The result is shown in Fig. A.10 on the right. Note that list T_e for edge e is drawn near the start vertex of e for space reasons. So there are entries from different edges in the list of one vertex (e.g., vertex v_2). All flow units arrive at vertex t up to time 17. By the previous lemma this is an upper bound for the flow value in the deterministic queuing model. So the flow value is maximum. ◁

With the lemma above we get the following result:

Theorem 16. *Let f^T be a maximum temporally repeated flow with time horizon T. Then Algorithm 1 transformed f^T into a maximum flow in the deterministic queuing model in polynomial time.*

Proof. Since Lemma 15 holds for the terminal vertex t, the flow value of the flow constructed by the algorithm (which gives the distribution of the flow to the edges for every vertex) is exactly the same as the flow value of the temporally repeated flow which was used as an input. Hence, starting with a maximum temporally repeated flow will give a flow with flow value equal to the capacity of a dynamic cut. Since this is an upper bound for the flow value in the deterministic queuing model, the resulting flow is maximum in this model.

In total we have to consider every vertex v twice for every path traversing it (when it becomes active and inactive again). Since there are at most m paths in the path decomposition for the temporally repeated flow in total we have at most $\mathcal{O}(m\,n)$ operations on the lists T_e for $e \in E$. If we store every list by means of a balanced data structure we can insert and remove entries in $\mathcal{O}(\log n)$ time. This yields a total complexity of $\mathcal{O}(m\,n \log n)$. □

Appendix B. Negative results in the sum-case

Here we summarize the negative results for the sum objective mentioned in the paper.

B.1. Example without a Nash equilibrium

First we give an example of a game with sum-objective, which has no Nash equilibrium. Note that this is only a slight modification from Example 2, which had no Nash equilibrium in the game with bottleneck objective.

Table A.1
Data for Example 5, for more details see text.

t_α	P^j	X_{pj}	f_{pj}	u	θ	$\widetilde{\theta}$	add to T_{e_i}	Y_{pj}	e_{pj}	add to $T_{e_{pj}}$	W_{pj}	$\widetilde{F}_{pj}$
Edge e_1, paths P^j, $j \in \{1\}$, buffer $W_{p1} = 44$, flow $\widetilde{F}_{p1} = 44$												
0	P^1	0	4	5	11	8	$(8, 0_{p1})$	5	e_4	$(2, 5_{p1})$	4	4
8	P^1	0	4	4	3	1	$(9, 0_{p1})$	4	e_4	$(10, 4_{p1})$	0	0
End of iteration; add $(11, 0_{p1})$ to T_{e_4}												
Edge e_2, paths P^j, $j \in \{2, 3\}$, buffer $W_{p2} = 50$, $W_{p3} = 33$, flow $\widetilde{F}_{p2} = 50$, $\widetilde{F}_{p3} = 33$												
0	P^2	0	5	18	10	4	$(4, 0_{p2}, 0_{p3})$	10	e_6	$(3, 10_{p2})$	10	10
	P^3	0	3					8	e_5	$(3, 8_{p3})$	1	1
4	P^2	0	5	11	6	1	$(5, 0_{p2}, 0_{p3})$	10	e_6	$(7, 10_{p2})$	0	0
	P^3	0	1					1	e_5	$(7, 1_{p3})$	0	0
End of iteration; add $(8, 0_{p2})$ to T_{e_6} and $(8, 0_{p3})$ to T_{e_5}												
Edge e_4, paths P^j, $j \in \{1\}$, buffer $W_{p1} = 0$, flow $\widetilde{F}_{p1} = 44$												
2	P^1	5	4	4	8	8		4	e_7	$(6, 4_{p1})$	8	12
10	P^1	4	4	4	1	1		4	e_7		8	8
11	P^1	0	4	4	2	2		4	e_7		0	0
End of iteration; add $(17, 0_{p1})$ to T_{e_7}												
Edge e_5, paths P^j, $j \in \{3\}$, buffer $W_{p3} = 0$, flow $\widetilde{F}_{p3} = 33$												
3	P^3	8	3	8	4	4		8	e_7	$(6, 8_{p3})$	0	1
7	P^3	1	1	1	1	1		1	e_7	$(10, 1_{p3})$	0	0
End of iteration; add $(11, 0_{p3})$ to T_{e_7}												
Edge e_6, paths P^j, $j \in \{2\}$, buffer $W_{p2} = 0$, flow $\widetilde{F}_{p2} = 50$												
3	P^2	10	5	5	5	5		5	t	$(8, 5_{p2})$	25	25
8	P^2	0	5	5	5	5		5	t	$(13, 5_{p2})$	0	0
End of iteration; add $(14, 0_{p2})$ to T_t												
Edge e_7, paths P^j, $j \in \{1, 3\}$, buffer $W_{p1} = W_{p3} = 0$, flow $\widetilde{F}_{p1} = 44$, $\widetilde{F}_{p3} = 33$												
6	P^1	4	4	7	4	4		4	t	$(7, 10_{p2})$	0	28
	P^3	8	3					3	t	$(7, 8_{p3})$	20	21
10	P^1	4	4	7	1	1		4			0	24
	P^3	1	3					3			18	18
11	P^1	4	4	7	6	6		4			0	0
	P^3	0	0					3			0	0
End of iteration; add $(18, 0_{p1}, 0_{p3})$ to T_t												

Example 6. Consider the graph $G = (V, E)$ from Fig. B.11 with $k = 6$ weighted users and the travel times and capacities as given in the figure near the edges as an instance of a dynamic routing game with sum objective. Commodities are denoted by (s_i, t_i) for user $i \in \{1, 2, 3, 4, 5, 6\}$ with priority i, where the first two users are the main users, while the other four users are auxiliary users.

Note that the auxiliary users from commodity 3 to 6 do not have a choice on which route to take. Their purpose is to slow down the other users. So they are called blocking users. The users from commodities 1 and 2 have two path choices each, because the outer paths have a capacity smaller than their weight. Then there are four strategy profiles left which are drawn as graphs in the right of Fig. B.11.

Whenever both of the main users choose their left paths (upper left strategy profile), then the lower user arrives at vertex v_4 (filled) at the same time as the left blocking user from commodity 3. Since the blocking users has an inferior priority, the blocking user is slowed down by one time unit. This is enough to reach vertex v_1 of the upper main user's path at the same time (4) as the blocking user from commodity 5 does. Since the edge (v_1, v_2) has capacity 2 and the total load entering the edge at time 4 is given by 3, one of the users arrives at vertex v_2 at time 5 and the other one at time 6. The main user from commodity 1 arrives at vertex v_1 at time 5, i.e., one time unit after the other two users. Hence, these users are allowed to leave edge (v_1, v_2) before her. Since user 1 has weight 2 equal to the capacity of the edge, she arrives at vertex v_2 one time step after the two blocking users left the edge, i.e., at time 7. So, she arrives at vertex v at time 8 and at the terminal at time 9, while the main user from commodity 2 reaches both vertices one unit in time earlier.

Hence, the user from commodity 1 changes to the upper right path (upper right strategy profile). Here, the blocking users arrive at vertex v_5 at the times 3 and 4 and at vertex v_6 at times 4 and 5, while the main user arrives at vertex v_5 at time 5 and at vertex v_6 at time 6 without being affected by the blocking users. So, both main users arrive at vertex v at time 7. Since the user from commodity 1 has a higher priority, she arrives at the terminal $t_{1,2}$ at time 8, one time unit earlier than in her old path. So, this change is an improvement for her, while the travel time of the main user from commodity 2 is increased by one time unit. Hence, this user is unsatisfied with her path and changes to the lower right path (lower right strategy profile), which results in the same case as when both users took the left paths. Continuing with this behaviour shows that there is no Nash equilibrium for this instance.

The analysis above also shows that there is always a path of length 8 for both of the main users and there is never a path with smaller travel time. Hence, we can assume that there is always a path for the user 1 with travel time 8 and never a path with a smaller travel time. If we remove the user from commodity 1, then every path of user 2 has the minimum travel time and, hence, gives a Nash equilibrium. We need these statements later in the proof of Theorem 19. ◁

B.2. *Hardness of* Quickest Flow

Here we show that the Quickest Flow problem is hard to solve with a reduction similar to the one in the proof of Theorem 1 in the bottleneck case.

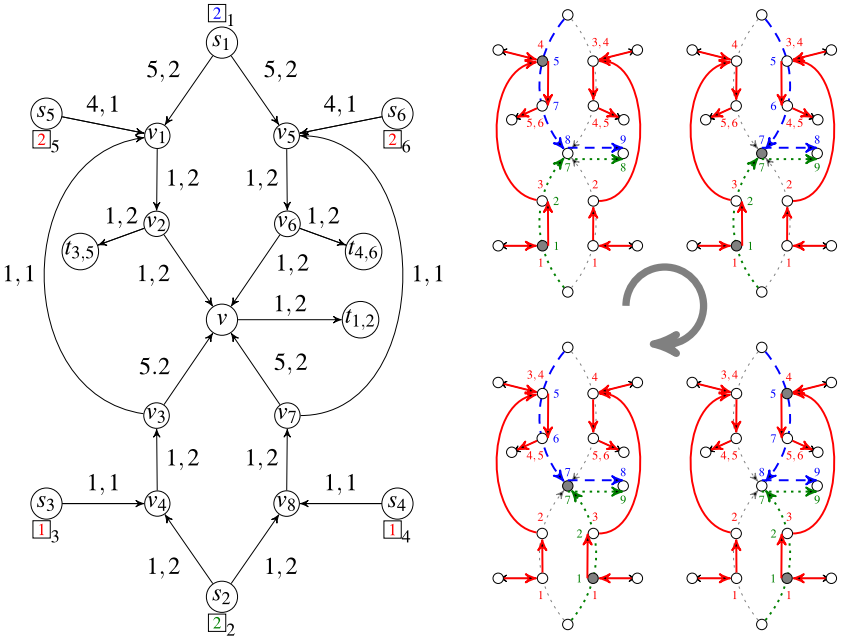


Fig. B.11. Left: instance with multi commodities which does not possess a Nash equilibrium. The first number near an edge denotes its travel time and the second one its capacity. The boxed expressions are the users and the index next to the box the priority of the user. Right: four relevant strategy profiles. Here, dashed (blue) lines are used for the first commodity, dotted (green) lines for the second commodity and solid (red) lines for the other four; unused edges are slightly grey. The numbers near some vertices show the arrival times of the users at these vertices. The inner numbers correspond to the main users, the outer ones to the auxiliary users. The filled vertices show situations where users enter an edge at the same time.

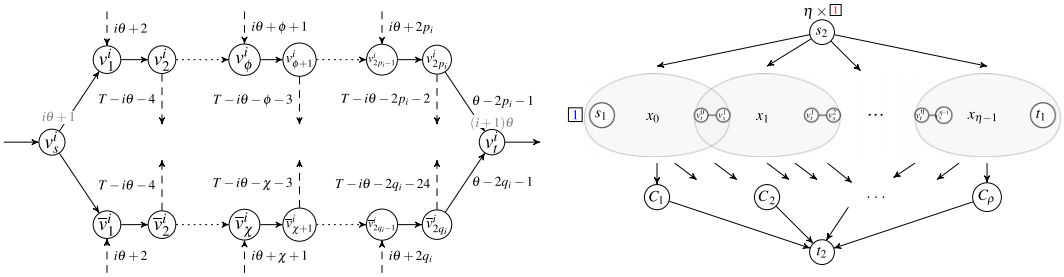


Fig. B.12. Illustration of the reduction from 3-SAT to the Quickest Flow problem in the two commodity case. Left: the graph shows a gadget representing a literal x_i, $i \in \{0, \ldots, \eta - 1\}$. The expressions near some of the edges are the travel times of these edges. The expressions near some of the vertices denote the expected arrival times at these vertices, if there is a feasible truth assignment. All capacities are set to 1. Right: shows the whole graph. The greyly shaded areas illustrate the gadgets.

Theorem 17. *The* Quickest Flow *problem is* NP*-complete in the strong sense in the two commodity case, even if all users are unweighted.*

Proof. We use the same reduction from 3-SAT as for the bottleneck case. The instance of 3-SAT is given by a set of Boolean variables $X = \{x_0, \ldots, x_{\eta-1}\}$, and ρ clauses $(C_j)_{j=1,\ldots,\rho}$ containing exactly three literals.

We construct an instance of Quickest Flow as we did for NarrowestFlow in Theorem 1. While the definitions are not repeated here, the illustration can be seen in the right part of Fig. B.12, which is nearly the same as in the bottleneck case. One difference is, that the edges with travel times greater than one do not have to be replaced by consecutive edges with travel time equal to one here. Furthermore, we do not need local priorities for the edges for the makespan objective.

Note that in a flow with makespan $\leq T$, the edges connecting the gadgets can only be used by the first user. Hence, any user from the second commodity can only traverse a single gadget. Since every clause is connected to the second terminal vertex t_2 by a single edge and a total amount of ρ users have to be routed for the second commodity, a flow with makespan $\leq T$ has to route a single unit of flow along each clause vertex. Furthermore, no user in this flow can be slowed down by another user, since every path has capacity 1, free travel time T and all users traversing a subpath of the path arrive at the common vertices at the same time.

With these observations it is easy to show that there is a flow with makespan $\leq T$ iff there is a feasible truth assignment.

Let f be a feasible flow with makespan T. Then, a single unit of flow routes from s_1 to t_1 without meeting another user. We set the variable x_i to true if this unit routes in the ith gadget along the lower path and we set the variable x_j to false if it routes in the jth gadget along the upper path. Then the users from the second commodity route along the vertices from the upper path in gadget i and along vertices from the lower path in gadget j. From these vertices one unit of flow is routed to each of the clause vertices and afterwards to the sink. Since at least one vertex in the path of a gadget of the corresponding variable is connected to a clause it is contained in. In any clause there is one vertex that is set to true by the flow. So, the flow induces a truth assignment.

On the other hand, given a truth assignment, we route the user from the first commodity in the ith gadget along the lower path, if the variable $x_i, i \in I$, in the truth assignment is true and we route the user in the jth gadget along the upper path, if the variable $x_j, j \in J$, is false. At least one vertex in this path inside a gadget of the corresponding variable is connected to a clause the variable is

contained in. In any clause there is one variable that is set to true. So, for the second commodity we route one unit of flow along a shortest path along a vertex in the upper path of each gadget $i \in I$ or one unit along a shortest path along a vertex in the lower path of each gadget $j \in J$ towards each clause vertex to which no unit of flow from the second commodity was already routed to. This gives a feasible flow and no two units of flow use the same edge. Hence, the makespan of this flow is given by T. □

The following lemma only states technical facts from the previous proof that are needed to prove Theorem 19.

Lemma 18. *For the instance built in the proof of Theorem 17 the following statements hold true:*

- *There is a flow for all k users (one user for commodity 1 and the remaining users for commodity 2) with makespan value equal to T iff there is a feasible truth assignment for the underlying 3-SAT problem. Otherwise, the makespan value is at least $T + 1$.*
- *If we allocate the worst priority to the user from the first commodity and the makespan of the flow is larger than T, then the travel time of the user from the first commodity is at least $T + 1$.*
- *All paths have travel time at least T, even if the number of users is changed.*
- *If the single user from commodity 1 is removed, then there is always a flow with travel time T, which can be computed easily.*

Proof. Results (1) and (3) follow directly from the proof above.

For the second statement we only have to note that if there is no feasible truth assignment, then the path of the user from the first commodity always intersects with the path of a user from the second commodity at the same time. Since the user from the first commodity has the lower priority, her travel time is increased by one and, hence, it is at least $T + 1$.

For the last statement consider the instance built in the proof of Theorem 17 with only the users from commodity 2. Then no edge is blocked by the user from commodity 1 any more. The only goal is to send one user of commodity 2 to each of the ρ clause vertices. Since every clause contains one of the literals in the gadgets we just pick one out of each. Since there is a path from source s_2 to each of these literals, we just route one user to this literal, then to the corresponding clause vertex and from there to the terminal t_2. This gives a flow for the $k-1$ users of commodity 2 with makespan value T, which is computed efficiently. □

B.3. Hardness of NashExistence

Here we show that the question whether a Nash equilibrium exists or not is hard to answer with a reduction similar to the one in the bottleneck case in the proof of Theorem 5.

Theorem 19 (NashExistence *is* NP*-hard*). *The* NashExistence *problem is* NP*-hard in the strong sense in the multi-commodity case, even if the graph is acyclic.*

Proof. Now the hardness is shown with a reduction similar to the one for the bottleneck case. Nevertheless we state the full construction here, because some travel times and capacities differ. An instance of 3-SAT is given by a set of Boolean variables $X = \{x_0, \ldots, x_{\eta-1}\}$, and ρ clauses $(C_j)_{j=1,\ldots,\rho}$ containing exactly three literals. The idea behind the subsequent construction follows from two former results:

We already know by Lemma 18 that the graph G_D constructed in the proof of Theorem 17, with one user routing for the first commodity, can be traversed with a travel time of T iff the instance of 3-SAT has a feasible truth assignment. Otherwise, the travel time is at least $T + 1$. On the other hand, without this user from the first commodity, the remaining $k - 1$ users can always route through the graph with makespan value T (independently on the instance of the 3-SAT problem).

Furthermore, we have seen in Example 6 an instance with a graph G_N, which has no Nash equilibrium if there is a single user routing for its first commodity. Additionally, there is always a path with travel time T for this user and never a path with smaller travel time. Without this user every feasible strategy profile of the other users gives a Nash equilibrium.

We add both graphs in one supergraph. First we add a super source s and connect it to the source s_1 of the first commodity in both graphs via an edge with travel time 1. We add a super terminal t and connect the terminal of the first commodity in G_N to this terminal via an edge with travel time 1. The capacities of all of these edges are set to 2. Next, we have to make sure that the user from the first commodity in G_D always suffers (i.e., has a makespan value of $T+6$) if the flow has a makespan value higher than $T+5$. To that end we need to extend the instance a little bit, by adding vertices s'_3, s'_4, t'_2, u, w and t'_1 and the following edges:

Edge	(s'_3, t_2)	(s'_4, t_2)	(t_2, t'_2)	(t'_2, w)	(t_1, u)	(u, w)	(w, t'_1)	(t'_1, t)
Travel time	T	$T+3$	1	2	1	2	1	1
Capacity	1	2	2ρ	1	2	2	2	2

The former users from commodity 2 in G_D now route from s_2 to t'_2 and are called clause users. We add an additional blocking user, who routes from s'_3 to t'_1 and one from s'_4 to t'_1. The users from the first commodity in both graphs are merged to one user, who routes from the new super source s to the new super terminal t and is called the main user here. Note that she has weight 2. All other users from graph G_N have the same sources and terminals as before. The priorities of the users are chosen in such a way that the main user has highest priority, the user from the second commodity in G_N has second highest priority, the blocking users in this subgraph have priorities 3 to 6, the clause users have priorities 7 to $6+\eta$, and the two remaining blocking users from s'_3 and s'_4 have priorities $7 + \eta$ and $8 + \eta$, respectively. The actual values of the priorities are chosen arbitrarily. The whole illustration is shown in Fig. B.13.

Claim 1. We have seen in Example 6, that in the graph G_N, there is always a path for the user from commodity 1 with makespan value 8. Hence, there is always a path from s to t with makespan value $T + 9$. But, then the user from the second commodity in this graph is not satisfied with her path.

Claim 2. The user from commodity 1 never routes along t_2, since the capacity of edge (t'_2, w) is 1 and her weight is 2.

Claim 3. Iff the flow of the clause users in the subgraph G_D has makespan value $\geq T + 1$, i.e., the main user arrives at vertex t_1 after time $T+1$ or a user from commodity 2 arrives at vertex t_2 after time T, then the user from commodity 1 cannot reach the terminal vertex t before time $T + 10$.

Proof. In the graph G_D with the modifications made above, all clause users arrive at vertex t_2 at time T, if none of them is slowed down before. At this time the user from s'_3 also arrives at vertex t_2. Since she has a lower priority, she has to wait for one time unit. Then she arrives at vertex w at time $T + 4$. The user from source s'_4 arrives at vertex w at time $T + 3$. The main user also traverses vertex w when routing through G_D. If she is not slowed down by any other user, then she arrives at vertex w at time $T+4$. Since she has the best priority, she traverses the subsequent edge first at this point in time and arrives at the terminal t at time $T + 9$.

If the main user was slowed down inside the graph G_D, then she arrives at vertex t_1 after time $T + 1$. Then she reaches the terminal after time $T + 9$.

Otherwise, the main user is not slowed down inside G_D and, if at least one of the clause users arrives at least one time unit later than

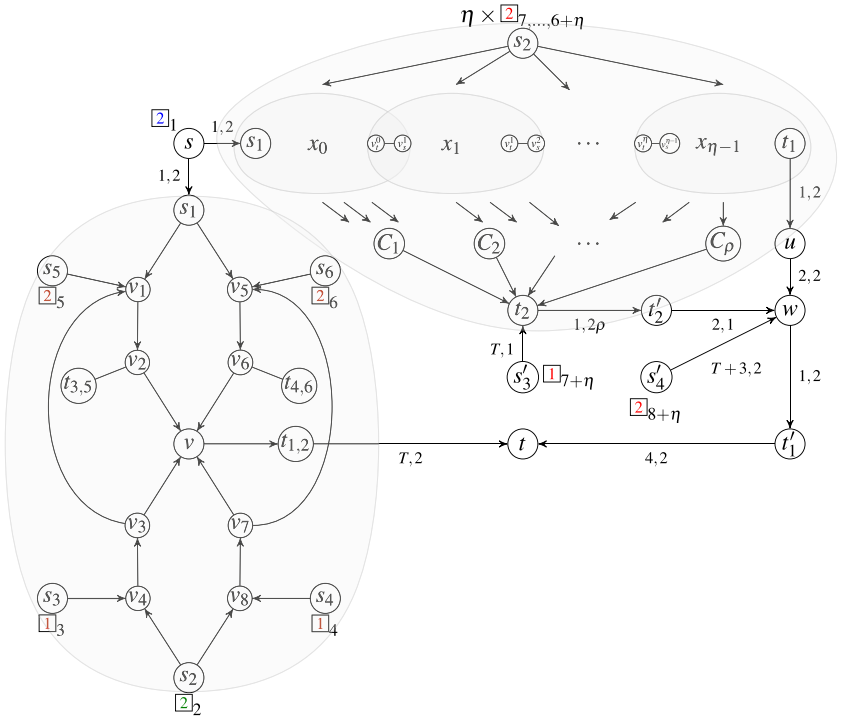


Fig. B.13. Illustration of the reduction from 3-SAT to NashExistence. The first expression near an edge is its travel time, the second one its capacity. The upper graph is from the reduction from 3-SAT to Quickest Flow used in the proof of Theorem 17, the left graph from Example 6. The exact travel times and capacities for the edges in these subgraphs can be found in the corresponding sections.

T at t_2, then the user from s'_3 arrives at vertex w at time $T+3$. The user from source s'_4 also arrives at vertex w at time $T+3$. If both users arrive at the same time, then the user from source s'_4 has to wait for one time unit in the waiting queue. The main user arrives at vertex w at time $T+4$. Since she enters edge (w, t'_1) after the other two users, she leaves the edge after them, i.e., not before time $T+6$. Then she enters the terminal not before time $T+10$. ◁

Main Claim. There is a feasible truth assignment for the 3-SAT instance iff there is Nash equilibrium in the enhanced instance.

Proof. Now assume that there is a feasible truth assignment for the 3-SAT instance. By Theorem 17 we can route the user from commodity 1 along a path from s to t_1 with travel time T. The clause users can also be routed with travel time T to the vertex t_2. Since they have a better priority than the blocking user from s'_3, they reach their terminal t'_2 at time $T+1$, which is best possible. By Claim 3 the main user traverses the edge (w, t'_1) with travel time 1, since the blocking user from source s'_3 is slowed down by the clause users. Hence, the main user arrives at her terminal t at time $T+9$, which is best possible. Without the user from the first commodity routing through the graph G_N, there is also a Nash equilibrium in this graph. Hence, there is a Nash equilibrium for the enhanced instance.

Assume conversely that there is no feasible truth assignment for the 3-SAT instance. By Theorem 17 any flow with one user routing for the first commodity in the graph G_D has makespan value $\geq T+1$. By Claim 3 the main user receives makespan value $\geq T+10$ up to the terminal t, when routing through the subgraph G_D. Since there is always a path in G_N with travel time $T+9$ by Claim 1, she changes her path to this graph. By Example 6, there is no Nash equilibrium in this subgraph. Hence, there cannot be a Nash equilibrium in the enhanced instance. □

References

[1] Rosenthal RW. A class of games possessing a pure-strategy nash equilibrium. Internat J Game Theory 1973;2(1):65–7.
[2] Roughgarden T, Tardos E. How bad is selfish routing? J ACM 2002;49(2):236–59.
[3] Nash JF. Equilibrium points in n-person games. Proc Natl Acad Sci 1950;36:48–9.
[4] Aumann RY. Acceptable points in general cooperative n-person games. In: Tucker AW, Luce RD, editors. Contributions to the theory of games IV. Annals of mathematics study, vol. 40. Princeton University Press; 1959. p. 287–324.
[5] Pigou AC. The economics of welfare. London: Macmillan; 1920.
[6] Koutsoupias E, Papadimitriou CH. Worst-case equilibria. In: Proceedings of the 16th international symposium on theoretical aspects of computer science. LNCS, vol. 1563. Springer; 1999. p. 404–13.
[7] Papadimitriou CH. Algorithms, games, and the Internet. In: Proceedings of the 33rd annual ACM symposium on theory of computing. New York (NY, USA): ACM; 2001. p. 749–53.
[8] Roughgarden T. Selfish routing and the price of anarchy. OPTIMA 2007;74.
[9] Skutella M. An introduction to network flows over time. In: Research trends in combinatorial optimization. Springer; 2009. p. 451–82.
[10] Köhler E, Möhring RH, Skutella M. Traffic networks and flows over time. In: Algorithmics of large and complex networks. 2009. p. 166–96.
[11] Koch R, Skutella M. Nash equilibria and the price of anarchy for flows over time. Theory Comput Syst 2011;49(1):71–97.
[12] Banner R, Orda A. Bottleneck routing games in communication networks. IEEE J Sel Areas Commun 2007;25(6):1173–9.
[13] European Union. Council Regulation (EC) No 1/2005 of 22 December 2004 on the protection of animals during transport and related operations and amending Directives 64/432/EEC and 93/119/EC and Regulation (EC) No 1255/97.
[14] Hill A, Neri F. Optical switching networks: from circuits to packets. Commun Mag 2001;39(3):107–8.
[15] Dorren HJS, Hill MT, Liu Y, Calabretta N, Srivatsa A, Huijskens FM, et al. Optical packet switching and buffering by using all-optical signal processing methods. J Lightwave Technol 2003;21:2–12.
[16] Baert E, Develder C, Colle D, de Turck F, Pickavet M, Demeester P. Routing strategies to minimize packet loss in an optical packet switched network with recirculating fdl buffers. Photonic Netw Commun 2004;7(3):279–300.
[17] Mukherjee Biswanath. Optical WDM networks (Optical networks). Secaucus (NJ, USA): Springer-Verlag New York, Inc.; 2006.
[18] Anshelevich E, Ukkusuri S. Equilibria in dynamic selfish routing. In: Proceedings of the 2nd international symposium on algorithmic game theory. LNCS, vol. 5814. 2009. p. 171–82.
[19] Farzad B, Olver N, Vetta A. A priority-based model of routing. J Theor Comput Sci 2008;1:1–29.
[20] Hoefer M, Mirrokni V, Röglin H, Teng S-H. Competitive routing over time. Theoret Comput Sci 2011;412(39):5420–32.
[21] Harks T, Klimm M, Möhring RH. Strong equilibria in games with the lexicographical improvement property. Internat J Game Theory 2013;42(2):461–82.
[22] Sperber H. Complexity of equilibria in games on networks, Part II: strong equilibria in Bottleneck and congestion games [Ph.D. Thesis], TU Kaiserslautern; 2009.

[23] Caragiannis I, Galdi C, Kaklamanis C. Network load games. In: Proceedings of the 16th annual international symposium on algorithms and computation. LNCS, vol. 3827. Springer; 2005. p. 809–18.
[24] Harks T, Hoefer M, Klimm M, Skopalik A. Computing pure nash and strong equilibria in Bottleneck congestion games. Math Program Ser A 2013;141: 193–215.
[25] Fotakis D, Kontogiannis S, Koutsoupias E, Mavronicolas M, Spirakis P. The structure and complexity of nash equilibria for a selfish routing game. Theoret Comput Sci 2009;410(36):3305–26.
[26] Correa JR, Schulz AS, Stier-Moses NE. Fast, fair and efficient flows in networks. Oper Res 2007;55(2):215–25.
[27] Busch C, Magdon-Ismail M. Atomic routing games on maximum congestion. Theoret Comput Sci 2009;410(36):3337–47.
[28] Werth TL, Sperber H, Krumke SO. Computation of equilibria and the price of anarchy in Bottleneck congestion games. CEJOR Cent Eur J Oper Res 2013.
[29] Even S., Itai A., Shamir A. On the complexity of time table and multi-commodity flow problems. In: 16th annual symposium on foundations of computer science. 1975, pp. 691–703.
[30] Garey MR, Johnson DS. Computers and intractability (a guide to the theory of NP-completeness). New York: W.H. Freeman and Company; 1979.
[31] Fotakis D, Kontogiannis S, Spirakis P. Symmetry in network congestion games: pure equilibria and anarchy cost. In: Proceedings of the 3rd international workshop on approximation and online algorithms. vol. 3879. Springer; 2005. p. 161–75.
[32] Ford LR, Fulkerson DR. Flows in networks. Princeton University Press; 1962.
[33] Ahuja RK, Magnanti TL, Orlin JB. Network flows: theory, algorithms and applications. Prentice-Hall; 1993.
[34] Schrijver A. Combinatorial optimization. Springer; 2003.
[35] Nemhauser GL, Wolsey LA. Integer and combinatorial optimization. Wiley-interscience series in discrete mathematics and optimization, John Wiley & Sons; 1999.