Quickest Flows: A practical model

Ekkehard Köhler
Brandenburgische Technische Universität Cottbus (Germany),
ekoehler@math.tu-cottbus.de

Rolf H. Möhring, Ines Spenke[*]
Technische Universität Berlin (Germany),
{moehring, spenke}@math.tu-berlin.de

March 14, 2008

Abstract

We consider quickest flows within a new model that is based on transportation applications. In contrast to other models, it is forbidden to store flow in nodes and to cross nodes with more than one flow unit simultaneously. We work on undirected graphs. Grid graphs are of special interest because they typically arise in practice. Our model allows to close edges temporarily by time windows, and considers waiting on edges.

We solve several quickest s, t–flow problems without time windows polynomially. We prove that time windows make these problems NP–hard and even not approximable. In a multicommodity environment, all quickest flow variants are shown to be NP–hard even in grid graphs with uniform edge transit times. An alternative proof shows NP–hardness already for a small number of commodities in case that waiting is not allowed and transit times are edge–specific. Finally, we propose two approximation algorithms in case that time windows do not occur.

1 Introduction

Modeling transportation problems inherently requires to reflect a time dimension. Transportation vehicles, e.g., in storage areas move along lanes within some time and leave them such that others can follow.

Ford and Fulkerson [6] formalize the concept of traveling flow to so–called *flows over time*. We consider the problem of finding a flow over time that sends a given amount of flow from source to sink nodes within a minimum possible time horizon called the *quickest flow problem.*

[*]supported by the DFG Research Center MATHEON "Mathematics for key technologies."

Our flow model reflects a lot of properties and behaviour coming from practice such as node capacities, waiting policies, or temporarily closed edges. Due to such properties, even well–established techniques, like time–expanded networks, are of little help. We concentrate our analysis on the complexity of a variety of problems and, if applicable, polynomial solution methods or approximation algorithms. Grid graphs are of special interest because they often arise in transportation applications.

Related work

Flows over time: Ford and Fulkerson [6] introduce flows over time: In a directed graph, flow enters arcs at discrete time steps respecting a capacity and a transit time function, both integral. Flow arriving at a node can be sent immediately along an incident arc or can be stored. Flow conservation has to be fulfilled: At each point in time, the flow leaving a non–terminal node does not exceed the flow that has arrived there until then. Furthermore, no flow remains in the graph after some time horizon. As for the static case, they find maximum s, t–flows over time within a given time horizon. Their algorithm builds so–called temporally repeated flows with no individual edge flow at each time step but instead sending the same amount of flow within a time period such that non–polynomiality of a full time expansion is avoided.

The quickest flow problem asks for the minimum time horizon to send a given amount of flow. It equates the maximum flow over time problem combined with binary search and, thus, is polynomially solvable in the single commodity case. Burkard et al. [1] solve it in strongly polynomial time using a parametric search method that is due to Megiddo [12].

Continuous flows over time are defined by an inflow rate into an arc at each moment of time. For these flows time–expanded graphs cannot be applied since there is no time discretization. Fleischer and Tardos [5] relate the two models to each other and show that many results hold for both.

Hoppe and Tardos [9] consider multicommodity flows over time and introduce polynomial algorithms for several problems, e.g., for the quickest transshipment problem. The results are based on a new class of dynamic flows called chain–decomposable flows, which again allow the repeat of flow.

Hall et al. [8] show that it is NP–hard to find a feasible multicommodity flow over time that satisfies demands within a time horizon. Thus, the multicommodity max flow over time problem is NP–hard for their model.

Fleischer and Skutella [4] work on quickest multicommodity flows. They give a 2–approximation based on the computation of static length–bounded paths. Adding a cost bound, a $2 + \epsilon$–approximation can be derived.

Disjoint paths: Our quickest flow problems are related to disjoint paths problems over time due to unit edge capacities and integral flow values. In contrast to the static disjoint paths problem (e.g. [14]), the dynamic version is not well–studied. Busch et al. [3] consider a routing problem where packets should be sent along fixed paths. Two packets are not allowed to share an edge in the same direction at the same time. The task is to send the packets such that they all arrive in minimum time without collisions. For this NP–hard problem a greedy method yields an approximation algorithm.

Flows and paths in grids: Kramer et al. [11] show NP–completeness of the disjoint paths problem even for rectangular grids. A survey of disjoint paths problems in grids is given by Kaufmann and Mehlhorn [10].

Burkard et al. [2] consider an orientation problem: In an uncapacitated, undirected graph with edge lengths and demands they ask for a min cost strong edge orientation fulfilling each demand along one shortest oriented path. Restricted to grids they describe a polynomial solution.

Rabani [13] considers path coloring in grids. A list of node pairs has to be connected by colored paths, where paths of the same color are edge-disjoint. He shows that minimizing the number of colors is NP–hard and describes an approximation based on LP techniques and randomized rounding.

Our model

We study discrete quickest flow problems. The intention of our model is to reflect typical behaviour of transportation applications, as, for example, the routing of automated guided vehicles in a container terminal. Conventional models in the literature mentioned above are often inconsistent with such practical properties. We describe and motivate our model in the following:

We work on undirected graphs following the idea that transportation lanes can be entered from both sides.

Edge transit times are integral, do not change over time and do not depend on the direction in which an edge is traversed. Two settings of either uniform transit times or edge individual transit times are considered.

Edges shall behave like lanes. Hence, edge capacities are set to 1: At most one unit of flow enters an edge at one time step no matter from which end node. We forbid flow of more than one unit to cross along an edge. Flow portions do not change their direction while traversing an edge.

Flow portions shall model vehicles and are, therefore, assumed to be integral. Thus, requiring an edge capacity of 1, either no flow or a flow of value 1 enters an edge at each time step.

We regard nodes as crossings with space for only one flow unit such that only one unit can traverse a node at each time step. It is not seen as favorable to wait at a crossing; thus, we forbid it in nodes but, however, allow it on edges for an integral duration. Waiting flow units cannot be crossed by other units neither in the same nor in the other direction. Flow can start to wait at each time step, also having already traveled along an edge for some time.

Technical reasons or formerly sent flow may block some edges for certain time intervals. We model such restrictions by means of so–called time windows that close edges temporarily.

Discrete models mentioned in the literature overview allow to build time–expanded networks. These networks, containing a copy of the entire graph for each time step, transform flows over time into static flows such that well–studied techniques can be applied on a pseudo–polynomial sized new graph. It is open whether our model properties can be reflected within a time–expansion or not. It is especially not clear how to transform undirected edges into directed ones such that edge capacities and crossing policies are still respected (see Spenke [15]). Hence, pseudo–polynomiality is not known.

Outline

Section 2 investigates flows over time with a single commodity and Section 3 with more than one commodity. We study different problems, varying by three criteria: Is waiting on edges allowed or not?, Can edges be temporarily closed by time windows or not?, Are transit times uniform or edge individual?

2 The single commodity case

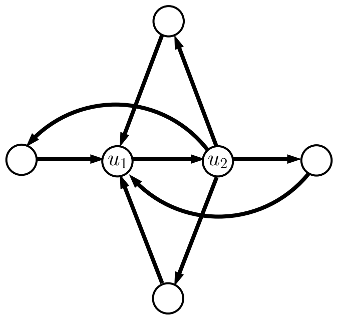
We start with the single commodity case and show in the following that quickest flows according to our model can be found polynomially if time windows on edges are not allowed, and that time windows cause NP–hardness and limit even the approximability of the problem.

2.1 No time windows on edges - Polynomial solvability

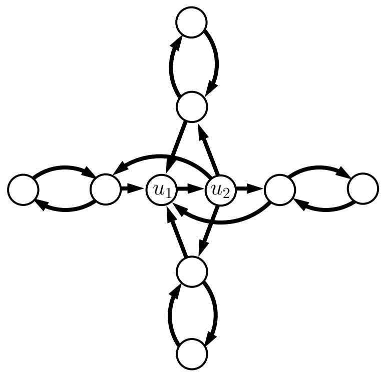
Ford and Fulkerson [6] introduce an algorithm that finds a maximum s,t–flow over time for a given time horizon in polynomial time. Combined with binary search this algorithm solves the quickest s,t–flow problem as well.

These two statements hold for their model but not obviously for ours. Ford and Fulkersons strategy relies on time–expanded networks: The nodes of the original directed network are copied for each time step and arcs are introduced between layers according to their transit times. Holdover arcs allow waiting of flow in nodes. Ford and Fulkerson determine a static flow in the original directed network, decompose it into path flows, and send flow along these paths repeatedly through the time–expansion. They get a maximum s,t–flow because of a 1-to-1 relation between an s,t–flow over time in the original graph and a static flow in its time–expansion.

It is open whether a time–expanded graph can be built reflecting all properties of our model. Especially the transition from undirected to directed graphs is still open (see details in [15]). Although such a transition is not known, we show that Ford and Fulkersons algorithm can be applied to find a quickest s,t–flow for our model if there are no time windows on edges.



(a) A node u is replaced by six nodes and nine arcs with transit time 0 and capacity 1.



(b) A node u is replaced as in (a) and the four adjacent undirected edges with pairs of directed ones.

Figure 1: At most one unit of flow is allowed to cross a node at one time.

Adapting Ford and Fulkerson's algorithm

Two modifications are needed before Ford and Fulkerson's algorithm can be applied to our model. First, to get a directed graph, each edge uv is replaced by two arcs (u, v) and (v, u), both with unit capacity and the transit time $\tau(uv)$ of the edge uv. Even though there is no explicit coupling between (u, v) and (v, u), we will see that capacity constraints can still be met.

Ford and Fulkerson allow arbitrarily large flow values in each node at any time. We ensure that each node is traversed by at most one flow unit per time step by splitting each node into a subnetwork with nine arcs and six nodes, see Figure 1 (a). All artificial arcs get zero transit time and unit capacity. Now all paths that traverse a node u in the original graph at a time t have to traverse the new arc (u_1, u_2) at time t. This arc's only unit capacity then avoids more than one flow unit traversing it at one time, such that at most one flow unit crosses u in the original graph at that time. Both modifications let arc and node numbers increase by constant factors only.

Now we can apply Ford and Fulkerson's algorithm and no flow crosses in nodes of the original graph. We get an s, t-flow over time which can be forced to send only integral flow portions along paths because of integral capacities. Their algorithm never uses the allowed waiting of flow in nodes.

There is one challenge left that we have to eliminate. If we replace each undirected edge with two opposing arcs, then, perhaps, flow is sent along an arc (u, v) and along its counterpart (v, u) starting at the same time. This is only possible if the static flow calculated by Ford and Fulkerson's algorithm sends a flow unit along (u, v) and another unit along (v, u). Such cycles can be cancelled out before decomposing the static flow into paths, see Figure 2.

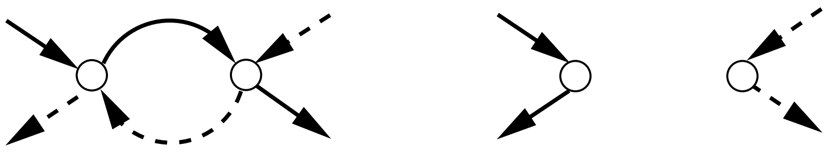


Figure 2: Left: A dashed and a solid path traverse opposing arcs. Right: The flow on these arcs is removed and the paths are recombined.

Proof of optimality

The obtained s, t-flow over time is a maximum flow despite model variances. Ford and Fulkerson's model allows the flow to wait in nodes. This is never applied with their algorithm, which gives the additional insight that waiting in nodes never yields a better maximum flow. Thus, applying their algorithm will never force our flow to wait in nodes, which would violate our model. On the other hand, we consider the case that waiting on edges is allowed. The described algorithm does not use waiting on edges. In the following, we show that allowing flow units to wait would not yield better solutions.

We assume to have computed a maximum flow f^t for our model that is allowed to wait on edges in the modified graph. This flow can easily be transformed into a flow according to Ford and Fulkerson's rules. If flow waiting on an edge is simply pushed into the head node of the according edge and let to wait there, the underlying graph and the flow value $|f^t|$ will still be the same. The only difference is that the flow now waits in nodes. Hence, we have a feasible flow that possibly waits in nodes. Applying Ford and Fulkerson's algorithm to the graph would compute a maximum flow over time $\bar{f}^t$ that is allowed to wait in nodes, hence, satisfying $|\bar{f}^t| \geq |f^t|$. Ford and Fulkerson have shown that their flow never uses waiting, thus, the flow $\bar{f}^t$ is valid for our model, too. Moreover, it does not wait on edges. The maximality of f^t and the feasibility of $\bar{f}^t$ in our model and the relation $|\bar{f}^t| \geq |f^t|$ imply that $\bar{f}^t$ is a maximum flow in our model, too, without waiting. Hence, allowing to wait on edges does not produce better flows.

The main result of this section is the following:

Theorem 1. *The quickest s, t-flow problem without time windows can be solved in polynomial time. Waiting on edges does not allow smaller time horizons.*

Proof. We have seen that we can calculate a maximum s, t-flow over time given a time horizon T in polynomial time, satisfying our model assumptions with Ford and Fulkerson's algorithm after making some polynomial graph modifications. This result, combined with binary search, solves the quickest s, t–flow problem. $\square$

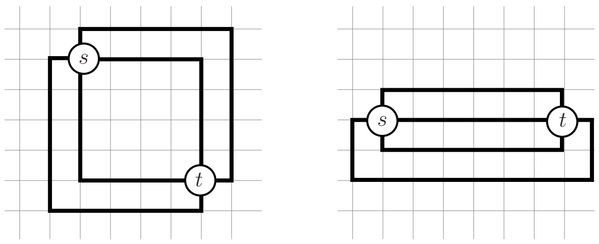


Figure 3: Optimal paths in a uniform edge transit time grid. (left: Source and sink not on the same grid line, right: Source and sink on one grid line)

In case of grid graphs with uniform transit times a quickest s, t–flow can be determined much easier: Start flow units along four predefined paths as it is shown in Figure 3 until the demand is sent. Flow starts along the shorter paths slightly longer such that all paths are finished nearly at the same time. Detailed start times and proofs of optimality can be read in Spenke [15].

2.2 Time windows on edges - NP–hardness

In this section, we allow time windows on edges. We assume that these time windows forbid entering an edge for certain time intervals. Hence, the network changes within time, such that we cannot suppose that predefined paths or temporally repeated paths as in the previous section, are successful.

Proof of NP–hardness

Gawrilow, Köhler, Möhring, and Stenzel [7] show that the quickest flow problem is polynomially solvable for a demand of 1 if there are time windows and if waiting on edges is allowed. The complexity for higher demand values remains open. We show that this problem becomes hard if waiting is not allowed even for a demand of 1 and even on grid graphs.

Theorem 2. *The quickest flow problem is NP–hard with time windows and if waiting on edges is not allowed, even for a demand of 1 and even on grids.*

Proof. We reduce from PARTITION to a quickest flow problem with demand 1, with time windows, and without waiting in a grid, see Figure 4.

Let us consider a PARTITION problem: Given positive integers $a_1, ..., a_r$, is there a partition of them into two groups with the same sum of elements? Let b be some integer with $b > \sum_{i=1}^{r} a_i/12$. We consider a grid graph as in Figure 4 and describe all required edges. All other grid edges are assumed to have a large transit time of M, such that they are not used during a later introduced time horizon. The nodes s and t are connected by horizontal edges with transit times $a_1 + 3b, ..., a_r + 3b$. Furthermore, for each of these edges,

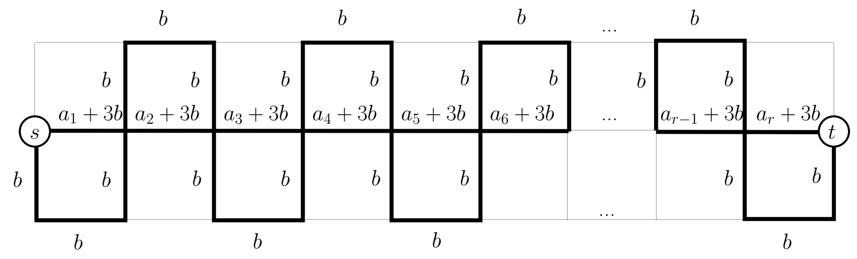


Figure 4: Reduction from PARTITION. Edge transit times of thick edges are denoted, thin edges get large transit times. All edge capacities are 1.

there is a three edge detour of transit time $3b$. We introduce time windows on four edges. The thick edges incident to s are closed by time windows $[1, M]$, which means that they can be entered at time 0 or later than M. Flow is not intended to reach t before time $\sum_{i=1}^{r} a_i/2 + 3br$. This is enforced by closing the vertical thick edge incident to t for time $[0, \sum_{i=1}^{r} a_i/2 + 3br - b]$ and the horizontal edge for time $[0, \sum_{i=1}^{r} a_i/2 + 3br - a_r - 3b]$. Both intervals are well–defined due to the non–negativity of their right bounds for all $r > 2$.

We consider the following quickest flow problem in this graph: Is it possible to send one flow unit without waiting within a time horizon of $\sum_{i=1}^{r} a_i/2 + 3br$ from s to t? We show that the answer is "yes" if and only if the PARTITION instance is satisfiable.

First, let us assume to know a path sending one unit of flow from s to t without waiting that arrives within time $\sum_{i=1}^{r} a_i/2 + 3br$. Obviously, this path does not travel along edges with a transit time of M assuming M to be a very large number; $M = \sum_{i=1}^{r} a_i/2 + 3br + 1$ is sufficient. Hence, the path traverses only the remaining edges with a transit time of $a_i + 3b$ or b. The structure of the subgraph consisting of such edges implies that for each $i = 1, ..., r$, either the edge with transit time $a_i + 3b$ or the according detour with a transit time of $3b$ has to be traversed at least once. In fact, time is not sufficient to traverse such an edge or its detour twice because then, only considering the b's, they add up to at least $3br + 6b > 3br + \sum_{i=1}^{r} a_i/2$ due to the size of b such that the required duration of the path is exceeded. Thus, the path contains either an edge of time $a_i + 3b$ or a detour of time $3b$ for each $i = 1, ..., r$. We let $I \subseteq \{1, ..., r\}$ contain the indices of traversed edges with transit time $a_i + 3b$. The path then has a transit time of $\sum_{i \in I} a_i + 3br$.

Flow can start at s at time 0 only or from time $M + 1$ on, but the second is too late to finish in time. Hence, the flow starts at time 0. Flow can enter t at the earliest at time $\sum_{i=1}^{r} a_i/2 + 3br$, which is the supposed path's time horizon. Thus, it is finished exactly at that time. Due to the fact that waiting is not allowed, the path's transit time is $\sum_{i=1}^{r} a_i/2 + 3br$. We have calculated the path's transit time in two ways, thus, $\sum_{i \in I} a_i + 3br = \sum_{i=1}^{r} a_i/2 + 3br$. Thus, the PARTITION instance is satisfied by choosing all $a_i \in I$.

Second, we suppose to have a satisfiable PARTITION instance with an index set I of satisfying elements. A flow unit that starts at time 0 along an s, t-path and traverses all edges of transit times $a_i + 3b$, $i \in I$, and travels along the detours of transit time $3b$ for all other i is finished without waiting at time $\sum_{i \in I} a_i + 3br = \sum_{i=1}^{r} a_i/2 + 3br$. Hence, the required flow exist. □

The proof fails if waiting is allowed because the path's transit time may then differ from the difference of the path's start and end times. It fails as well in case of uniform transit times. In both cases the complexity is open.

Non–approximability

Lemma 3. *Consider a quickest flow problem with time windows. Let T_{opt} be an optimal time horizon, and let α and β be functions that are polynomially in the problem's input size. Each polynomial approximation with a guarantee of $\alpha \cdot T_{opt} + \beta$ can be used to solve the problem exactly in polynomial time.*

Proof. Suppose that there is a polynomial algorithm A that calculates a flow with time horizon $T_A(I)$ and that satisfies for all instances I:

$$T_{opt}(I) \leq T_A(I) \leq \alpha \cdot T_{opt}(I) + \beta.$$

We show that the decision problem: *Is there an s, t-flow over time with value d and time horizon T?*, can be solved by A in polynomial time. This implies the polynomiality of the optimization problem.

Let us consider an arbitrary quickest flow instance J and a time horizon T. By means of the algorithm A, we want to answer the question whether J can be solved within T or not. We transform J into an instance J' by adding time windows on all edges e incident to the sink t such that flow cannot be finished within time $[T+1, \alpha \cdot T + \beta]$. This is done by closing these edges to entering within the time interval $[T - \tau(e) + 1, \alpha \cdot T + \beta - \tau(e)]$.

We apply A to J'. According to the time windows on all edges incident to the sink, A calculates a flow that arrives at the sink either within $[0, T]$ or later than $\alpha \cdot T + \beta$. If A returns a flow finished up to time T, the same flow solves the instance J. Hence, J is finished up to time T, too. Otherwise, if the algorithm returns a time horizon for J' larger than $\alpha \cdot T + \beta$, then J' cannot be finished up to time T due to the assumed approximation guarantee. But then J as well cannot be finished up to time T because, otherwise, such a solution would solve J' within T. Hence, the following holds:

$$\begin{array}{lclcl} T_A(J') \leq T & \Rightarrow & T_{opt}(J') \leq T & \Rightarrow & T_{opt}(J) \leq T \text{ and} \\ T_A(J') \geq \alpha \cdot T + \beta + 1 & \Rightarrow & T_{opt}(J') > T & \Rightarrow & T_{opt}(J) > T. \end{array}$$

Thus, solving J' with the algorithm A within time T is equivalent to the fact that J is solvable in time T, which means that all instances of the above decision problem can be solved in polynomial time. □

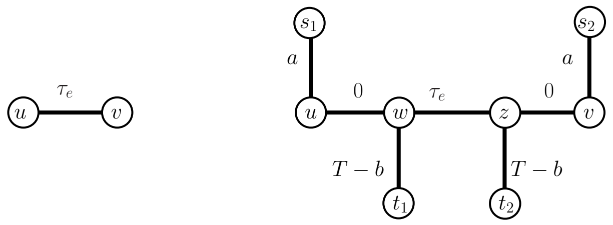


Figure 5: Replacing of a time window $[a, b]$ on an edge uv.

Lemma 3 yields that showing NP–hardness for a quickest flow problem with time windows implies a non–approximability for all guarantees α and β as described above, unless $P \neq NP$, already for a demand of 1.

3 The multicommodity case

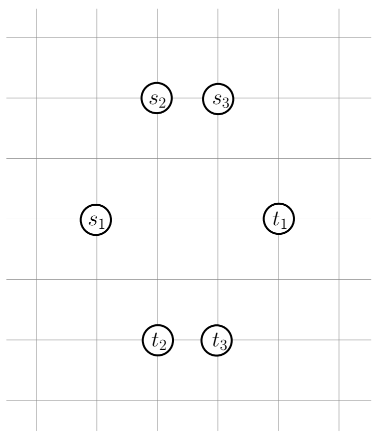
In this section, we consider multicommodity flows. We first differentiate this case from the single commodity one, show then NP–hardness for all problem variants and propose approximation algorithms.

3.1 Differences to the single commodity case

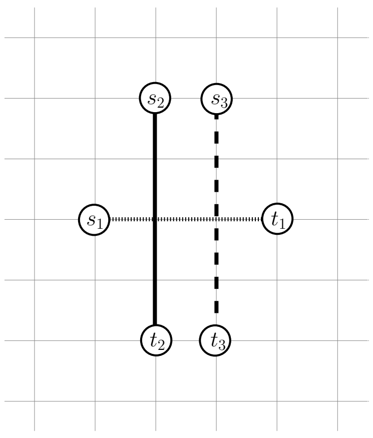
With only one commodity the existence of time windows is decisive for the problem's complexity. In a multicommodity environment, time windows do not seem to be the crucial point; they can be replaced by additional commodities whose sources, sinks, and demands force some edges to be blocked for a certain time. Such a construction is sketched in Figure 5: A time window $[a, b]$ forbids to enter an edge uv at times $\{a, a+1, ..., b\}$. This can be simulated by introducing two commodities from s_i to t_i, $i = 1, 2$, each with $b - a + 1$ demand. Transit times are given in the figure. The new edges get a capacity of 1. The two new commodities can only be satisfied if they block the new edges uw and zv within $[a, b]$. This replacement can be applied for more than one time window on an edge and also preserving a grid (see [15]).

In contrast to a single commodity flow a multicommodity one can profit from waiting even without time windows. In Figure 6 (a), an instance with three commodities is given, each with a unit demand. All edge transit times are 1. In both cases, with and without waiting, there is a quickest flow traveling along the direct source–sink paths shown in Figure 6 (b). If waiting is allowed, a quickest flow is finished at time 4, without waiting at time 5.

With only one commodity, we have seen that there is always a quickest flow with integral flow values if there are no time windows. We do not know if this is still true when time windows are present. However, if we allow several commodities, then fractional quickest flows can be quicker than



(a) A three commodity instance is given. All demands and edge transit times are 1.



(b) Optimal paths are drawn, valid for both cases, with and without waiting.

Figure 6: A quickest flow needs four time units with waiting and five without.

integral flows if time windows are introduced. In Figure 7 an example is sketched. We assume demands of d for each commodity and edge transit times of 1. Edges that we want to use are marked with thick lines; all thin grid edges are assumed to be closed by time windows within the time horizon. All source–sink paths cross pairwise at the same distance from their sources such that fulfilling the demands with integral flow portions can only be done successively. Assuming k commodities, demands are fulfilled not before time $k \cdot d$ plus the length of an s_i, t_i–path. Allowing fractional flow portions, each commodity can start half a unit of flow at each time step such that the flow is finished at time $2d$ plus the length of an s_i, t_i–path. Hence, by choosing, k, d and path lengths appropriately, the ratio between a quickest flow with fractional flow values to one with integral values is not bounded.

3.2 Proofs of NP–hardness

Hall, Hippler, and Skutella [8] show that the multicommodity flow over time problem is NP–hard with or without flow storage in nodes from two commodities on. Hence, this holds for maximum multicommodity flows over time and, thus, for quickest flows. We cannot adapt these results due to model differences. In contrast to us, they work with arbitrary edge capacities and fractional continous flows, and they allow arbitrary flows to cross in nodes.

We show by a reduction from 3-Coloring that the quickest flow problem in our model is strongly NP–hard if transit times are uniform, with and without time windows, with and without waiting, on grids. This implies NP–hardness for all considered problem variants even for uniform transit times, that are handled as transit times 1 on every edge, and for grids.

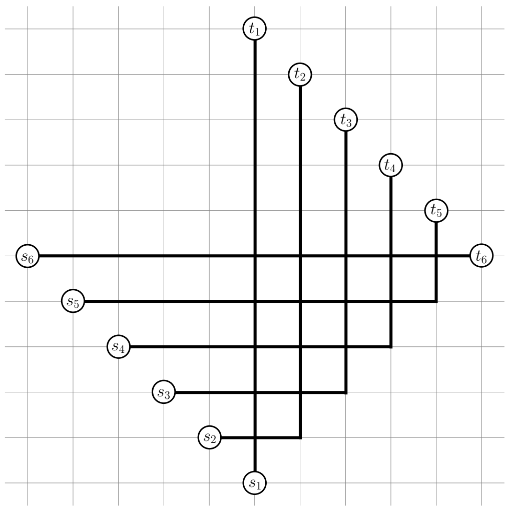


Figure 7: An instance is given in which a fractional quickest flow is arbitrarily faster than an integral one, chosing the number of commodities appropriately.

Afterwards, we give an alternative reduction from PARTITION. It holds if edge–specific transit times are allowed and in all variants of time windows or not and with waiting or not in grids. This second reduction permits to refine complexity results by identifying small numbers of commodities for which the problem is already hard.

3-COLORING

We reduce the strong NP–hard 3-COLORING problem:

Is it possible to color the nodes of a given graph with three colors, such that adjacent nodes have different colors?

to multicommodity quickest flow problems. First, we explain a basic version that holds for grids if transit times are uniform, if time windows are introduced, and if waiting is not allowed. Afterwards, we replace time windows with additional commodities and show that waiting can be allowed. The idea of the reduction is similar to an approach in Busch et al. [3].

Theorem 4. *The quickest multicommodity flow problem with time windows, without waiting, and with uniform transit times is strongly NP–hard in grids.*

Proof. To facilitate the readability of the proof, we apply a coloring instance, see Figure 8. In Figure 12, a grid graph transforming it into a flow problem is shown whose construction and use is explained in the following.

The basic idea of the flow graph is shown in Figure 9. For each node of the graph to color, $G = (V, E)$, a source s_i and a sink t_i are introduced. Time

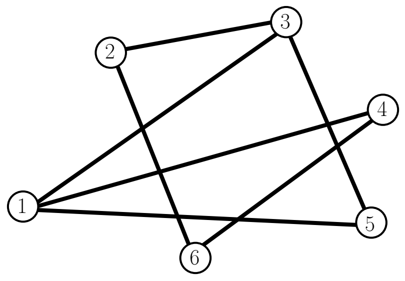


Figure 8: An instance of the 3-Coloring problem with six nodes

windows, demands, and a time horizon will be set in such a way that the required flow can only be satisfied along predefined paths that are marked in the figure by different line styles. These paths cross each other pairwise. At each crossing, both involved paths have the same distance to their sources; thus, simultaneously started flow units along two of the paths cause a collision. We replace each of these crossings by gadgets, such that paths belonging to non–adjacent nodes in G can be started simultaneously without causing a collision, and such that paths of adjacent nodes still collide. Hence, two different gadgets are used. A first gadget provides a detour for the crossing paths, such that a collision can be avoided no matter at which time flow units start along the paths. It is inserted exactly at the crossings of an s_i, t_i-path and an s_j, t_j-path if i and j are not adjacent in G. Otherwise, if i and j are adjacent in G, then another gadget is introduced with two–fold consequences. On the one hand, simultaneously started flow units still collide. On the other hand, with these gadgets, paths become longer, such that detours caused by the first gadgets are compensated. Thus, the property that two paths have the same distances from their sources to their crossing is maintained even if they have already passed some gadgets.

Consequently, flow along the predefined paths of two commodities can start simultaneously if and only if there is no edge joining the associated nodes in G. Exactly those nodes of G can be colored with the same color. We interpret a simultaneous start of path flows in the flow graph as coloring the nodes in the coloring graph with the same color. Therefore, three colors are sufficient to color G if and only if at most three starting times suffice to satisfy a demand of 1 for each commodity in the flow graph. Thus, we verify whether such demands can be sent starting only at the three time steps 0,1,2.

As suggested in the previous paragraph, some effort is needed to construct a graph transforming the coloring problem into a flow problem. We explain the construction in the following. First, we denote the position of terminal nodes. Subsequently, we introduce the gadgets, and, finally, we explain how to assure that only the predefined paths are used. The basic graph from Figure 9 is transformed into a graph $G' = (V', E')$, shown in Figure 12.

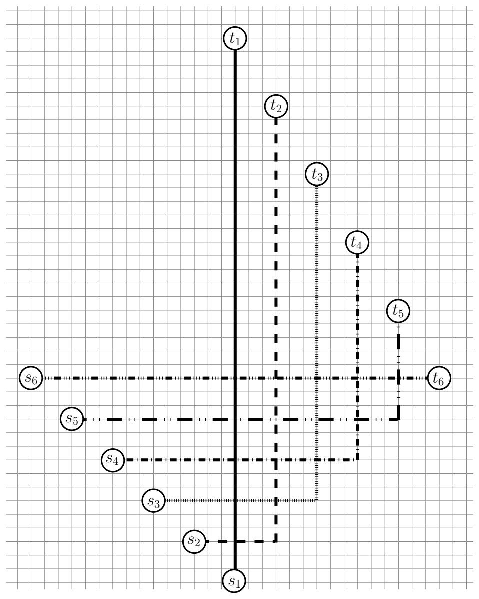


Figure 9: First step of a transformation of 3-Coloring to a quickest flow.

Terminal nodes: Assuming to have a grid graph we fix one node to be the source s_1. Considering the grid as a two–dimensional coordinate system, we define s_1 as the point of origin. Neighboring sources differ by 3 in both directions, so the sources are positioned at the coordinates $s_i = (-3(i-1), 3(i-1))$, for all $i = 1, ..., |V|$. The sinks have coordinates $t_i = (3(i-1), 8|V| - 3 - 5i)$, differing horizontally by 3 and vertically by 5.

Gadgets: As stated above, we want the commodities introduced so far to be satisfied along predefined paths. The flow of two commodities should be allowed to start along the paths exactly at the same time if the according nodes in the graph to color, G, are not adjacent. This is ensured by replacing the path's crossings with gadgets. Replacing does not mean adding edges. The grid provides us with all of the needed edges. It means instead making them available within the entire time horizon by applying no time windows to them, as we will later do for other edges.

Consider a crossing of two predefined paths, one from s_i to t_i and one from s_j to t_j, as is sketched in Figure 10 (a). Simultaneously started flow along these two paths collides at node u. In the case that $ij \notin E$, we replace situation (a) by introducing the gadget from Figure 10 (b). There are two nodes where the path flows may now cross. The s_j, t_j–path, which can choose between two traversings of the gadget, (c) and (d), arrives at both crossing nodes at the same time θ. The path from s_i to t_i reaches the possible crossing nodes at different times θ' and $\theta' + 2$, such that at least one of them avoids a collision, no matter at which time flow units along both paths start.

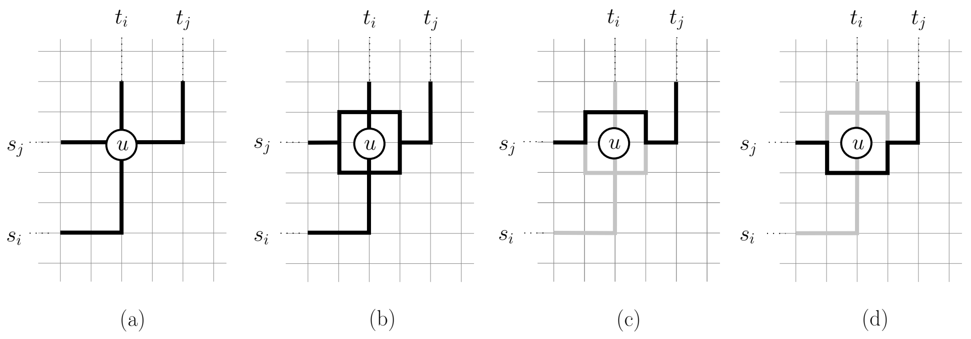


Figure 10: A gadget replaces each path crossing according to non–adjacent nodes. In (a), a crossing is sketched, in (b), its replacement, and in (c) and (d), the two possibilities to traverse it with a path from s_j to t_j.

Lemma 5. *Two predefined paths that belong to non–adjacent nodes in G can be used for transporting one flow unit each without collision, no matter at which time the flow units start.*

Now we consider two paths that are assigned to two adjacent nodes in G. We introduce the subnetwork from Figure 11 (b). The s_j, t_j–path has two possibilities to traverse the gadget, as is shown in (c) and (d). If both paths from s_i to t_i and from s_j to t_j start at the same time, then they meet at u, such that a collision occurs. If a flow unit starts earlier in s_i than in s_j, then the s_i, t_i–path shown in (c) avoids a collision. Otherwise, starting later, the path from (d) yields a feasible solution. One could ask why we have introduced these gadgets. Simultaneously started path flows collide as before. However, both gadgets cause a detour of the same length, namely being two additional edges, such that distances from the sources to the former crossings still have the same length for the crossing paths. Consequently, former conflicts still exist if paths have already traversed some gadgets.

Lemma 6. *Two predefined paths that belong to adjacent nodes in G can each be used for transporting one flow unit without collision if and only if they start at different times.*

Predefined paths: Finally, we have to ensure that the flow travels along the predefined paths including gadgets. First, we set some parameters that will be needed later. We let all edge transit times and all demands of previously–introduced commodities be 1. We then set the time horizon to $8(|V| - 1) + 2$. Second, we want to restrict an s_i, t_i–path from traveling vertically towards a sink that is not t_i, apart from single edges belonging to the path's detours. Therefore, we add commodities, which has the result that each vertical path can only be used by the first commodities for one unit of flow, thus, only by the commodity whose sink is reached along that vertical

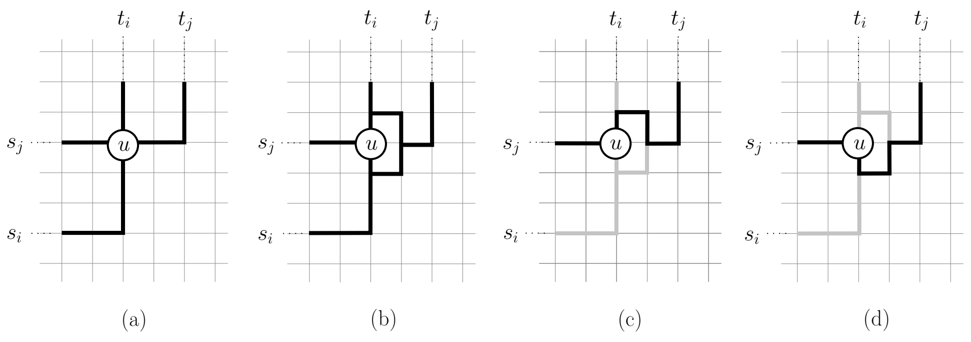


Figure 11: A gadget replaces each path crossing according to adjacent nodes. In (a), a crossing is sketched, in (b), its replacement, and in (c) and (d), the two possibilities to traverse it with a path from s_j to t_j.

path. These artificial commodities have sources $s_i = (3(i - |V| - 1), 0)$ and sinks $t_i = (3(i - |V| - 1), 3|V|)$ and a demand of $d_i := 4|V| + i - 5$, for $i = |V| + 1, ..., 2|V| - 1$. They are visualized in Figure 12. All grid edges that are not marked by thick lines in this figure must not be used. We close them with time windows within the entire time horizon of $[0, 8(|V| - 1) + 2]$.

All of the settings from the previous paragraph imply the following. According to time windows, flow can only be sent along the thick edges in Figure 12. We consider an artificial commodity with source $s_{|V|+i}$. Its source–sink distance is $3|V|$. Having the source and the sink on one vertical grid line, each $s_{|V|+i}, t_{|V|+i}$–path has to be crossed by $|V| - i$ flow units, satisfying the first commodities. A demand of $4|V| + i - 5$ can only be satisfied by traveling vertically straight at all times where the first commodities do not cause a collision because this strategy allows to transport $(8(|V| - 1) + 2) - 3|V| - (|V| - i) + 1 = 4|V| + i - 5$ units of flow to the sink within the entire time horizon. All other paths are too long to be used.

By blocking vertical edges within the time horizon apart from a few instances, the artificial commodities force the first commodities to travel along the predefined paths. That means that they first travel horizontally, apart from traversing some gadgets, until they are on one grid line with their sink and then travel vertically to the sinks.

The construction of the flow graph is now finished. Based on an instance of 3-Coloring, we assume to have built such a graph G' with commodities, transit times, a time horizon, demands, and time windows as described above. Now we shall explain how to make use of it for solving 3-Coloring by showing that the following holds:

G can be colored with three colors.	$\Leftrightarrow$	There is a flow in G' without waiting that satisfies all demands within the time horizon.

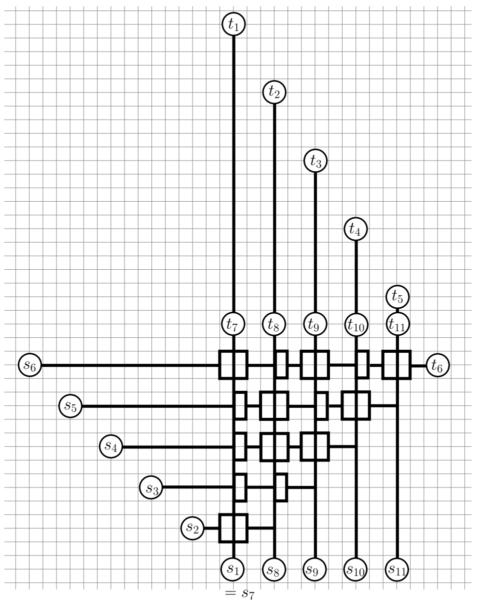
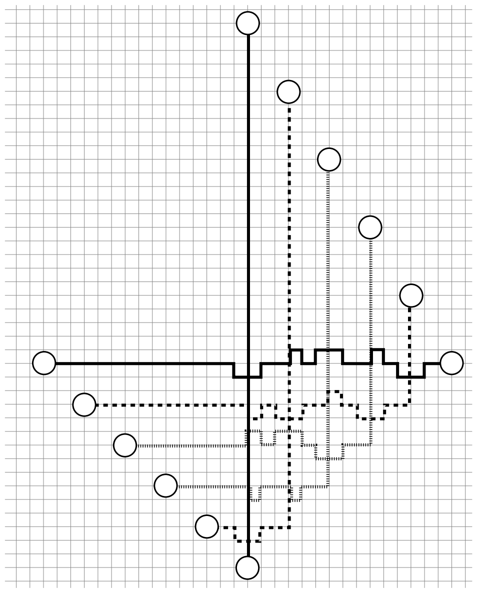


Figure 12: Transformation of a 3-Coloring instance to a quickest flow. All thin edges are closed by time windows for the entire time horizon.

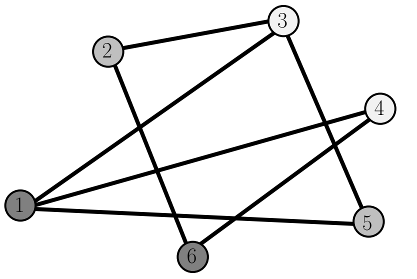
"⇒": Let us assume to have colored G with three colors 0,1,2. We will show that an appropriate flow can be sent in G'.

First, we describe along which paths flow units are sent. Let us consider all nodes that are colored with color 0. We start a flow unit along each of the associated paths at time 0. We let all flow units travel to the right and traverse gadgets until the flow unit is on one vertical grid line with the appropriate sink, which is then reached by continuing vertically. Lemma 5 proves that paths through gadgets can be chosen, such that no collisions arise. Now let us consider all nodes that are colored with color 1. We start flow again along the according paths but at time 1. For two of the path flows, collisions can be avoided by an appropriate choice of paths; see Lemma 5. For each pair of paths, one started at time 0 and the second at time 1, traversing the gadgets upwards or downwards has to be carefully chosen but can be done without collisions in case of adjacency, because of Lemma 6, and in case of non–adjacency, because of Lemma 5. In the same manner, flow along all paths associated to nodes colored with color 2 is started at time 2. Artificial commodities, not being associated with nodes of G, are satisfied vertically straight along s_i, t_i-paths. We send flow along these paths from time 0 on at all times where they do not collide with the first $|V|$ paths.

We show that the described flow satisfies all demands within the time horizon. By construction of G', all paths used to send one flow unit from s_i to t_i, $i \in \{1, ..., |V|\}$, have the same length $8(|V| - 1)$. When sending flow as



(a) Quickest flow solution: Differently marked paths start at different times.



(b) Solution to the 3-Coloring problem: The nodes 1 and 6 have the same color, as well as 2 and 5, and 3 and 4.

Figure 13: A quickest flow solution is presented and transformed into a solution of the coloring problem.

described, each of these paths finish at the latest at time $8(|V|-1)+2$. The demands of commodities $|V|+1, ..., 2|V|-1$ remain to be transported. As stated above, they are sent along the vertical paths between their sources and sinks. Each of these vertical paths is consumed exactly once by one of the first commodities from some edge on, namely by the commodity with the sink on the same vertical grid line. In addition, the vertical path of commodity $|V|+i$ is crossed by $|V|-i-1$ of the first commodities. Within a time horizon of $8(|V|-1)+2$, the commodity $|V|+i$ with a path length of $3|V|$ is finished $4|V|+i-5$ times, which equals the demand for all $i = 1, ..., |V|-1$.

Thus, all demands arrive at their sinks during the time horizon. The flow problem is solved.

"$\Leftarrow$": We assume to have a solution to the flow problem. Now we have to show how to produce a feasible coloring of G. In order to do so, we first analyze which paths are used by the flow.

As explained above, all paths for commodities from $|V|+1$ on have to travel purely vertically to be on time. The flow of the first $|V|$ commodities is satisfied along predefined paths.

A solution of the coloring problem can easily be derived. The first $|V|$ commodities travel along paths with a duration of $8(|V|-1)$. Due to a time horizon of $8(|V|-1)+2$, each of them has to start at times $0, 1, 2$, thus, at most at three different times. Choosing the same color for all nodes of G whose according s_i, t_i–flow starts simultaneously requires at most three colors. If there is an edge ij in G, then i and j have different colors because, otherwise, they would have the same starting time in G', causing a collision of the two paths at their crossing gadget.

In Figure 13, a flow solution is drawn according to the instance in Figure 8. Solid paths start at time 0, dashed paths at time 1, and dotted paths at time 2. This reflects a coloring with one color for the nodes 1 and 6, one for 2 and 5, and one for 3 and 4, as visualized in Figure 13.

The strong NP–hardness results from the fact that the numbers in the reduced flow instance are polynomially bounded in the input size of the 3-Coloring problem.

□

As explained in the beginning of this section time windows can be replaced by other commodities. We cannot use the replacement explained above if we want to preserve uniform edge transit times but rather an easier one because edges have to be closed for the entire time horizon and not for certain time intervals. Figure 14 visualizes the idea. For an entire time horizon of T, we close edges su, sv, sw, and sz by introducing four commodities, each with source s but with different sinks u, v, w, z and a demand of T for each of them. The demands can only be satisfied if s sends one flow unit along each incident edge at all time steps $0, 1, ..., T-1$. No matter along which paths these flows travel, the four edges incident to s are consumed by them within the entire time horizon closing them to other flow units. This holds for entering from both sides because of unit edge transit times. In spite of this easy approach, it is rather technical to describe the entire construction, for short we prevent the flow from using paths other than the predefined ones by introducing new commodities to all edges that are adjacent to such a predefined path but that do not belong to any of the paths. Furthermore, we add commodities to edges that can be used by paths for the amount of time that is not needed to send flow along them. The construction in full detail can be read in Spenke [15]. It results that Theorem 4 still holds if there are no time windows on edges.

Until now, the flow is not allowed to wait. It is easy to see, that in the above construction no flow unit can wait if all demands have to be fulfilled in time. Consequently, they do not wait, even if we allow it.

Partition

In the previous section we succeeded to formulate a reduction with uniform transit times. This reduction consumes a large number of commodities that depends on the number of nodes of the coloring graph. If we allow edge

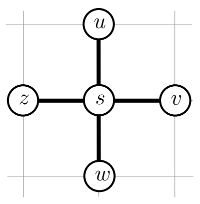


Figure 14: Time windows are simulated by additional commodities.

specific transit times then an alternative, much easier reduction of Partition shows NP–hardness, similar to Theorem 2. With this proof we lose the property of strongly NP–hardness but we get hardness already for small constant numbers of commodities.

Lemma 7. *It is NP-hard to find a quickest multicommodity flow without time windows and without waiting in grids with non–negative integral transit times. This already holds with two commodities.*

Proof. We use a similar graph to represent the Partition instance as in Theorem 2, see Figure 15. In contrast to the construction there, the first source and sink nodes are moved horizontally for two grid nodes each, a second pair of source and sink is introduced, and some edges get transit times of 1 instead of a large M. Time windows are removed; the second commodity will replace them. Consider the following decision problem:

Is it possible to send $d_1 = 1$ unit of flow from s_1 to t_1 and a demand of $d_2 = \sum_{i=1}^{r} a_i + 6br - 8 - 2r$ from s_2 to t_2 within time $\sum_{i=1}^{r} a_i/2 + 3br + 4$?

Let us first suppose that this question is answered with "yes". We show in the following that the demand d_2 has to be satisfied along the dashed and the dotted paths drawn in Figure 16. The dashed path has a transit time of $9 + r$. The dotted path has a transit time of $7 + r$. Sending one unit of flow along each of them from time 0 on as long as the time horizon is not exceeded means that the dashed path transports $\sum_{i=1}^{r} a_i/2 + 3br + 4 - 9 - r$ flow units and the dotted path $\sum_{i=1}^{r} a_i/2 + 3br + 4 - 7 - r$. This is, in total $\sum_{i=1}^{r} a_i + 6br - 8 - 2r = d_2$, such that d_2 is satisfied. A third possible path from s_2 to t_2 uses solid edges in the middle of the graph. A flow unit sent along such a path inhibits the starting of a flow unit along the dotted path at the same time and, additionally, a flow unit sent along the dashed path at a later time because of collisions either at the beginning or at the end of the path. Thus, two flow units from the former solution cannot be sent. But as seen before, we cannot afford to replace two flow units by only one if d_2 has to be fulfilled within the time horizon. Thus, only the dashed and the dotted paths are used for the second commodity.

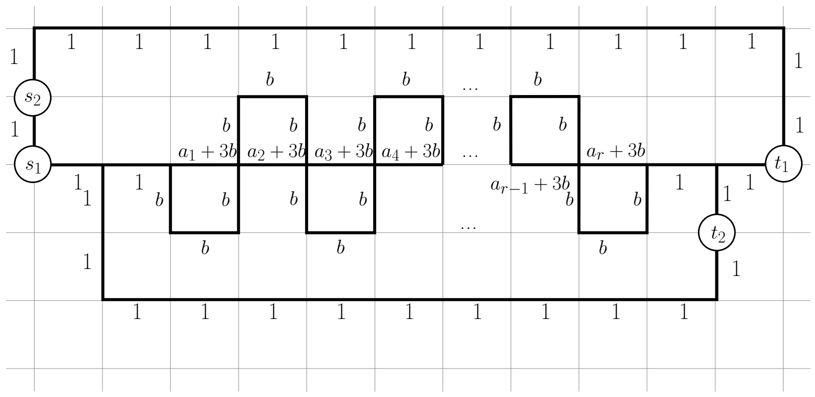


Figure 15: A grid graph is drawn to reduce Partition to a quickest flow problem with two commodities. Numbers represent edge transit times. All thin edges have a transit time of a large value M.

Furthermore, one flow unit has to be sent from s_1 to t_1. Having already consumed the dashed and the dotted paths within the entire time horizon, the path for d_1 has to travel along solid edges with one dotted edge at the beginning and one dashed at the end. The dotted edge is already used within a time interval of $[1, \sum_{i=1}^{r} a_i/2{+}3br{-}r{-}3]$ and the dashed edge within a time interval of $[7{+}r, \sum_{i=1}^{r} a_i/2+3br+4-9-r+7+r]$. The first commodity can only be satisfied within the time horizon if flow starts at time 0 and arrives at the last edge at time $\sum_{i=1}^{r} a_i/2{+}3br{+}4{-}9{-}r{+}7{+}r{+}1 = \sum_{i=1}^{r} a_i/2{+}3br{-}3$, which means to be at the sink at time $\sum_{i=1}^{r} a_i/2{+}3br{-}4$. Following the same argumentation as in Theorem 2, such a path implies that the Partition instance is satisfiable.

Secondly, we suppose to have a satisfied Partition instance. The decision problem is solved in an obvious way. As in Theorem 2, the first commodity is served by traversing solid edges. The second uses the dashed and the dotted paths as described above.

$\square$

The graph used in Lemma 7 can be modified by introducing more commodities, such that it holds as well if waiting is allowed. This requires a large number of additional commodities and provides, therefore, no more insight than the reduction of 3-Coloring. Thus, we do not go into detail about that. Notice that introducing time windows does not make the problems easier to solve from a complexity point of view. For the case with time windows, Lemma 7 implies the NP–hardness from two commodities on as well. But this is unnecessary to see because we already know the hardness from one commodity on in the case that was shown in Lemma 7.

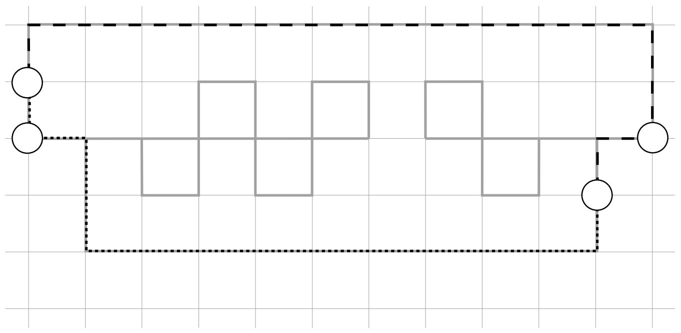


Figure 16: A dashed and a dotted path satisfy the second commodity.

Approximation

All single commodity problems with time windows cannot be approximated with reasonable guarantees; see Lemma 3. Thus, also for more than one commodity it remains to consider cases without time windows. Since all single commodity problems without time windows are polynomially solved we could solve the multicommodity problem consecutively. The overall flow time then equals at most k times an optimal time where k is the number of commodities, resulting in a k–approximation. We propose a second approximation that requires uniform transit times and a grid. It constructs flows without waiting, but still holds if waiting is allowed.

Description of the approximation. We use predefined paths, as in Section 2, in fact only one path for each commodity with the effect that possible collisions between paths of different commodities can be controlled.

We first partition the commodities into four groups according to their source–sink position, see Figure 17. We let, for example, group 1 contain all commodities with s_i on the left-hand side of t_i and s_i on the top or at the same height as t_i. Even if limited to only one group, the problem is strongly NP–hard since 3-Coloring only uses group 2 commodities.

Each commodity is served along one shortest path with at most one bend, see Figure 17. All paths start horizontally and continue vertically.

The groups and the commodities within them are routed in an arbitrary order. We assume that groups are routed in the order 1,2,3,4 without loss of generality. We suppose, furthermore, that the commodities are sorted, such that the first k_1 of them belong to group 1, the next k_2 to group 2, k_3 to group 3, and the last k_4 to group 4, with $k := k_1 + k_2 + k_3 + k_4$.

The first commodity is served along a shortest path as in Figure 17 starting flow units at times $0, ..., d_1 - 1$. Flow units for the second commodity travel along such a shortest path as soon as possible and until the demand is satisfied. If this path has a conflict with the path of commodity 1 for

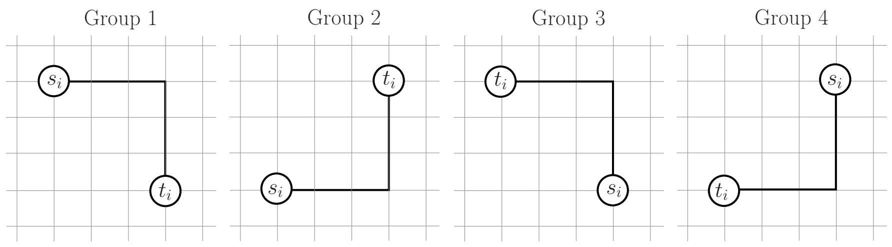


Figure 17: The commodities are partitioned into 4 groups depending on the positions of s_i, t_i. Shortest s_i, t_i–paths with at most one bend are sketched.

certain starting times, then we forgo starting the path for commodity 2 at those times. All following commodities within group 1 are routed in the same manner: Flow is sent along its path as soon as possible, only forgoing the start if a conflict with an already–sent commodity would arise.

When all of the first group's flow has reached the sinks, thus, no flow is left in the network, group 2 is routed with the same method. The commodities $k_1 + 1, ..., k_1 + k_2$ are sent consecutively, always forgoing starting times when they would cause collisions with already–sent flow from commodities of group 2. We proceed with group 3, followed by group 4 in the same manner.

Algorithm analysis. Two arbitrary paths in a grid can cross several times such that a flow unit sent along one path could prevent starting flow along the other for many time units. This is avoided by our predefined paths:

Lemma 8. *Consider two group 1 commodities with shortest paths p, q, each with at most one bend. Starting a flow unit along p at some time forbids starting along q for at most one time step.*

Proof. If p and q cross at all, then either in one node or sharing a path segment traversed in the same direction, see Figure 18. In both cases, a flow unit started along p forbids the starting along q for only one time. In the first case, both flow units must not be at the crossing at the same time, in the second their simultaneous entrance in the shared segment must be avoided. □

Lemma 9. *Group 1 is satisfied at time $\sum_{i=1}^{k_1} d_i + \max_{i=1,...,k_1} \ell_i - 1$, where d_i is the demand of commodity i and ℓ_i is the s_i, t_i–distance.*

Proof. Flow units for the first commodity start along a shortest s_1, t_1–path at times $0, ..., d_1 - 1$. Hence, this flow is finished at time $d_1 - 1 + l_1 = d_1 + l_1 - 1$. The second commodity cannot start for at most d_1 time steps (Lemma 8), because of a delay by each sent flow unit of the first commodity for at most one time step. Thus, its last unit of flow starts at the latest at time $d_1 + d_2 - 1$

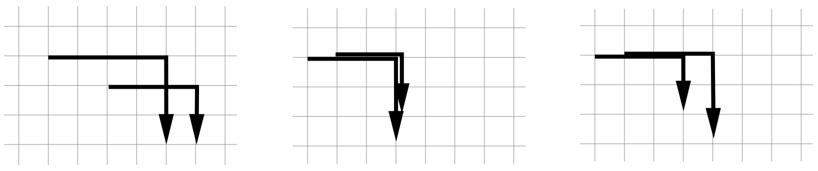


Figure 18: Two paths from one group can cross once or share one path segment that is traversed in the same direction.

and arrives at time $d_1 + d_2 + l_2 - 1$. The flow of a third commodity is delayed by at most $d_1 + d_2$ because of conflicts with the first two commodities. Its demand has left the source up until time $d_1 + d_2 + d_3 - 1$ and reached the sink up until time $d_1 + d_2 + d_3 + l_3 - 1$. Continuing, it follows that commodity i is started up until time $d_1 + ... + d_i - 1$ and is finished up until time $\sum_{j=1}^{i} d_j + \ell_i - 1$. So the flow of group 1 arrived at the sinks up until time:

$$\max_{i=1,\ldots,k_1} \{\sum_{j=1}^{i} d_j + \ell_i - 1\} \leq \sum_{i=1}^{k_1} d_i + \max_{i=1,\ldots,k_1} \ell_i - 1.$$

$\square$

We wait until the last flow unit of group 1 has arrived. Group 2 is started and finished by the same arguments as in Lemma 9, up until time:

$$\sum_{i=1}^{k_1} d_i + \max_{i=1,\ldots,k_1} \ell_i + \sum_{i=k_1+1}^{k_1+k_2} d_i + \max_{i=k_1+1,\ldots,k_1+k_2} \ell_i - 2 \leq \sum_{i=1}^{k_1+k_2} d_i + 2 \max_{i=1,\ldots,k_2} \ell_i - 2.$$

In the same manner, an upper bound for the overall flow time is:

Lemma 10. *The algorithm flow time has an upper bound of*

$$\sum_{i=1}^{k} d_i + 4 \cdot \max_{i=1,\ldots,k} \ell_i - 4.$$

The quickest flow time is bounded from below by $\max_{i=1,\ldots,k}(\lceil d_i/4 \rceil + \ell_i)$ in a grid: No algorithm is faster than starting four flow units at each time step and using only shortest paths. Thus, the algorithm has a guarantee of $4k$, which is of little help having a k–approximation by a sequential solution.

Nevertheless, a k–approximation means then a guarantee not better than $k(\max_{i=1,\ldots,k}(\lceil d_i/4 \rceil + \ell_i))$. Hence, the second approximation is better than sequential routing if the source–sink distances are large compared to the demands. This holds if, e.g., $\frac{3}{4} \sum_{i=1}^{k} d_i \leq \sum_{i=1}^{k}(\ell_i + 1) - 4 \cdot \max_{i=1,\ldots,k} \ell_i + 4$. This criterion can be verified directly from the input data.

Remark 11. *In comparison to a quickest flow lower bound the proposed strategy is not better than sequential routing. If source–sink distances are large in relation to the demands, the second method gives better guarantees.*

References

[1] R. E. Burkard, K. Dlaska, and B. Klinz. The quickest flow problem. *ZOR Methods and Models of Operations Research*, 37:31–58, 1993.

[2] R. E. Burkard, K. Feldbacher, B. Klinz, and G. J. Woeginger. Minimum–cost strong network orientation problems: Classification, complexity, and algorithms. *Networks*, 33:57–70, 1999.

[3] C. Busch, M. Magdon-Ismail, M. Mavronicolas, and P. G. Spirakis. Direct routing: Algorithms and complexity. In *Proceedings of the 12th European Symposium on Algorithms (ESA'04)*, volume 3221 of *LNCS*, pages 134–145. Springer Berlin, 2004.

[4] L. Fleischer and M. Skutella. The quickest multicommodity flow problem. In *Proceedings of the 9th International Conference on Integer Programming and Combinatorial Optimization (IPCO'02)*, volume 2337 of *LNCS*, pages 36–53. Springer Berlin, 2002.

[5] L. Fleischer and E. Tardos. Efficient continuous-time dynamic network flow algorithms. *Operations Research Letters*, 23:71–80, 1998.

[6] L. R. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.

[7] E. Gawrilow, E. Köhler, R. Möhring, and B. Stenzel. Conflict-free real-time AGV routing. *Operations Research Proceedings 2004*, pages 18–24, 2005.

[8] A. Hall, S. Hippler, and M. Skutella. Multicommodity flows over time: Efficient algorithms and complexity. *Automata, Languages and Programming*, 2719:397–409, 2003.

[9] B.E. Hoppe and E. Tardos. The quickest transshipment problem. In *Proceedings of the sixth ACM–SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 512–521. Society for Industrial and Applied Mathematics, 1995.

[10] M. Kaufmann and K. Mehlhorn. Routing problems in grid graphs. *Paths, Flows and VLSI–Layout*, pages 165–184, 1990.

[11] M.R. Kramer and J. van Leeuwen. The complexity of wire–routing and finding minimum area layouts for arbitrary VLSI circuits. *Advances in Computing Research*, 2:129–146, 1984.

[12] N. Meggido. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4:414–424, 1979.

[13] Yuval Rabani. Path coloring on the mesh. In *Proceedings of the 37th Symposium on Foundations of Computer Science (FOCS'96)*, pages 400–409. IEEE Computer Society, 1996.

[14] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.* Springer Berlin, 2003.

[15] I. Spenke. *Complexity and Approximation of Static k–Splittable Flows and Dynamic Grid Flows.* PhD thesis, Technische Universität Berlin, 2007.