Scheduling Transfers of Resources over Time: Towards Car-Sharing with Flexible Drop-Offs

Kateřina Böhmová[1], Yann Disser[2], Matúš Mihalák[1,3], and Rastislav Šrámek[4]

[1] Department of Computer Science, ETH Zurich, Switzerland
`katerina.boehmova@inf.ethz.ch`
[2] Department of Mathematics, TU Berlin, Germany
`disser@math.tu-berlin.de`
[3] Department of Knowledge Engineering, Maastricht University, The Netherlands
`matus.mihalak@maastrichtuniversity.nl`
[4] Google Zurich, Switzerland
`rastislav.sramek@gmail.com`

Abstract. We consider an offline car-sharing assignment problem with flexible drop-offs, in which n users (customers) present their driving demands, and the system aims to assign the cars, initially located at given locations, to maximize the number of satisfied users. Each driving demand specifies the pick-up location and the drop-off location, as well as the time interval in which the car will be used. If a user requests several driving demands, then she is satisfied only if *all* her demands are fulfilled. We show that minimizing the number of vehicles that are needed to fulfill *all* demands is solvable in polynomial time. If every user has exactly one demand, we show that for given number of cars at locations, maximizing the number of satisfied users is also solvable in polynomial time. We then study the problem with two locations A and B, and where every user has two demands: one demand for transfer from A to B, and one demand for transfer from B to A, not necessarily in this order. We show that maximizing the number of satisfied users is NP-hard, and even APX-hard, even if all the transfers take exactly the same time. On the other hand, if all the transfers are instantaneous, the problem is again solvable in polynomial time.

Keywords: Interval scheduling, complexity, algorithms, transfer, resources

1 Introduction

In car sharing services, a company manages a fleet of cars that are offered to customers for rent for a short period of time. Every car is stationed at a fixed parking location, and a customer who wishes to rent the car is usually required to return the car back to the very same location. This is a constraint that many customers would like to abolish. It is thus a natural question to find alternatives allowing the customers a *flexible* drop-off possibility. We investigate this "flexible drop-off" idea in the case where the demands for driving (pick-up at location A at time t_A and drop-off at location B at time t_B) are known in advance, and we study the problem of finding a maximum number of demands that can be realized by the existing fleet of cars and parking locations.

We show that the problem can be solved in polynomial time by a reduction to the minimum-cost maximum-flow problem in a dedicated auxiliary graph. We further consider the problem when every user (customer) has multiple driving demands. A user is satisfied if all her demands are fulfilled (the user needs to get a car for all the requested drivings, and has no interest in partial rentals). We show that satisfying the maximum number of users is an APX-hard problem already when there are only two locations, every user has two demands, the time for driving is the same for every demand, and there is only one car. An exemplary problem that falls into this setting is the situation where a single car is used to commute between two popular, but (by public transportation) badly connected locations. The users want to use this car for their daily travel: Every user wants to get from one location to the other one, and later in the day also from the other location back to her original one. Interestingly, the hardness holds only whenever the travelling takes non-zero time, as we also show that for an instantaneous travel (that takes zero time), the problem becomes solvable in polynomial time.

1.1 Formal Problem Description and Outline of the Paper

We define formally only the setting with two locations, as this setting forms the base of our main results. The problem definition for more locations is straightforward. User that rents a car at location A effectively blocks the car for a fixed time interval, and makes it available at the drop-off location B. The usage and trajectory of the car in the rental period is irrelevant for our scheduling problem, and we can simply model the renting as a *transfer* of the car from location A to location B at the given time interval. We abstract from our car-sharing motivation, and refer to the cars as resources.

We consider two locations, A and B, with an initial distribution of indistinguishable resources within these two locations, say there are a resources at location A and b resources at location B in the beginning. A transfer from A to B is a movement of a resource from A to B. A transfer is possible only if there is an available resource. There are n users and each of them has one or more demands: A demand d is specified by a direction $X \to Y$ (either $A \to B$ or $B \to A$) and time interval (t_d^s, t_d^e), and represents a request for a resource transfer from location X to location Y, leaving the origin X at time t_d^s and arriving at the destination Y at time t_d^e. The demand d is *fulfilled* by moving one resource from X to Y. In this case, the resource is blocked (i.e., cannot be transferred further) for the time period (t_d^s, t_d^e). The goal is to select a feasible set of demands that maximizes the number of satisfied users. Here, a set of demands is *feasible* if: (i) whenever a demand of a user is selected, then all demands of the user are selected, and (ii) all selected demands can be fulfilled, i.e., we can move the resources as suggested by the demands.

We first considered the simplest questions: (1) Decide whether all users can be satisfied (or equivalently, decide whether all the demands can be fulfilled);

(2) Compute the minimum number of resources initially needed at each location to satisfy all the users. We observed that straightforward "simulation-like" algorithms can answer these questions for any number of users, demands per user, locations, and resources.

In Section 2, we study the problem where each user has only one demand. We show that the problem of maximizing the number of satisfied users for given number of resources at locations (i.e., in this case, the number of fulfilled demands) is polynomially solvable, by reducing it to the minimum-cost maximum-flow problem. This approach works even if there are multiple locations and multiple resources in the system.

In Section 3, we study the variant where every user has exactly two demands: One transfer from A to B and one transfer from B to A, but not necessarily in this order. Recall that a user is satisfied only if both the demands are fulfilled. We show that in this setting, it is APX-hard to maximize the number of satisfied users even if (i) there is only one resource in the system, initially placed at location A, and (ii) all transfers take the same *non-zero* time (independently of the user and the direction). On the other hand, if the transfer time is always 0 (i.e., $t_d^s = t_d^e$ for every demand d), we show that this problem is polynomially solvable even if there are many resources in the system.

1.2 Related Work

Our problem lies in the area of interval scheduling (for recent surveys see [10, 11]), where, in the simplest case, one asks for a maximum non-intersecting subset of a given set of intervals. This simplest case would correspond to our setting with only one location A and every request of type "pick-up at A and drop-off at A".

In our problem with several locations, the transfers of a resource correspond to non-intersecting intervals (demands), with the following additional requirement: we label the interval with the corresponding pick-up and drop-off locations, and any two consecutive intervals for the same resource need to be compatible, i.e., the drop-off location of the first interval needs to be identical to the pick-up location of the second interval. For the setting with one resource and one demand per user, we ask for a maximum set of non-intersecting intervals with exactly this compatibility condition. With k resources (and one demand per user), we ask for k "chains" of such compatible solutions that together contain the maximum number of intervals (demands).

If every user has two or more demands, our problem relates to results on split intervals. A t-split interval is simply a union of t disjoint intervals. A t-interval graph is a conflict graph of n t-split intervals. Bar-Yehuda et al. [2] study the problem of finding the maximum number of non-intersecting t-split intervals, and show that it is APX-hard even when $t = 2$, and present a $2t$-approximation algorithm. Neither the approximation algorithm (or its techniques) nor the hardness result carries over to our problem with one resource and two locations. The main reason is that in our problem we require neighboring intervals in the solution to be compatible. These local compatibility requirements that we impose on the

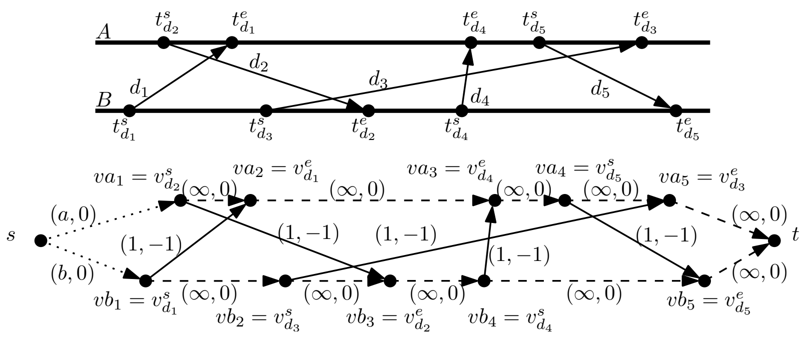


Fig. 1. An example of TransfersOneDemand transformed into minimum-cost maximum-flow problem. The labels on the edges in the second figure specify the capacity and the cost of the edges.

solution is what also makes our problem hard. As we will see, the hardness of our problem arises even in some configurations of intervals that would be trivially polynomially solvable under the split intervals setting (no local compatibility requirement). In particular, if every split interval intersects at most one other split interval, then the conflict graph forms a matching, and finding a maximum independent set becomes trivial. In our hardness result, in the reduction we use we obtain exactly such instances. The hardness arises due to the compatibility requirements.

Finding the maximum number of non-intersecting split intervals with certain additional pattern requirement has been studied before with the relation to problems in RNA secondary structure prediction. The 2-interval pattern problem (see e.g., [3, 7]) asks for a non-intersecting subset of 2-split intervals such that every pair of selected split intervals are in one of the prescribed relations $\mathcal{R} \subseteq \{<, \sqsubset, \between\}$ (with $<$ meaning preceding, $\sqsubset$ nested, and $\between$ crossed split intervals). The complexity as well as (approximation) algorithms for different subsets of $\{<, \sqsubset, \between\}$ were studied.

2 Resource Transfers with One Demand per User

If every user has only one demand (either of the form $A \to B$ or $B \to A$), and there are, initially, a resources at location A and b resources at location B, we show that TransfersOneDemand, the problem of maximizing the number of satisfied users (which is in this case equal to the number of fulfilled demands), is solvable in polynomial time.

Theorem 1. TransfersOneDemand *is solvable in polynomial time, for any number of locations.*

Proof. We formulate the problem as a minimum-cost maximum-flow problem, which is polynomial-time solvable. For simplicity, we present this reduction considering two locations (A and B) only. The generalization for arbitrary number of locations is straightforward.

Consider an arbitrary instance of TransfersOneDemand with two locations. We construct an instance of the network flow problem, where the only arcs of non-zero cost correspond to the demands of the users, and have cost -1. Formally, we proceed as follows (see Figure 1 for illustration). For every demand $d = (t_d^s, t_d^e)$ there are two vertices in the network, v_d^s and v_d^e, one for each endpoint of d. The network contains two additional vertices – a source s, and a target t. Based on the location of each demand's endpoint, the corresponding vertex is either of type A, or B. Let $\langle va_1, \ldots, va_n \rangle$ be the vertices of type A ordered by the time of the corresponding demands' endpoints, and let $\langle vb_1, \ldots, vb_n \rangle$ be the vertices of type B ordered by the time of the corresponding demands' endpoints. The network contains edges of three types. For every demand $d = (t_d^s, t_d^e)$, there is a directed *demand edge* (v_d^s, v_d^e) from the vertex corresponding to the start-point of d to the vertex corresponding to its endpoint. All demand edges have capacity 1 and cost -1. For every two consecutive vertices va_i, va_{i+1} of type A, there is a directed *connecting edge* (va_i, va_{i+1}). Similarly, there is a connecting edge for every two consecutive vertices of B. There are also connecting edges (va_n, t) and (vb_n, t), from the last vertices of each type to the target vertex. For all the connecting edges, the capacity is set to ∞ and the cost is set to 0. Finally, there is an edge from s to vertex va_1 with capacity a and cost 0, and there is an edge from s to vb_1 with capacity b and cost 0.

Observe that from any vertex other than s, there is unlimited capacity for a flow to t, using the connecting edges. Clearly, any st-flow has to pass via va_1 or vb_1, and the sum of capacities of (s, va_1) and (s, vb_1) is $a+b$. Thus, the maximum st-flow is of size $a + b$.

We now determine the cost of an optimum minimum-cost maximum st-flow. Since all the costs and capacities are integral, then, thanks to the integrality theorem [5], there exists an integral optimum solution (which can be found in polynomial time). From the above it follows that there is a maximum st-flow of cost 0 that does not use any demand edge. Since the network is acyclic, the capacity of a demand edge is 1, and its cost is -1, it follows that a minimum-cost st-flow aims at using as many demand edges as possible (with unit flow on each edge). We can see the integral flow as the course of the $a + b$ resources between the locations – one st-flow of size one per resource. Obviously, an integral st-flow of cost $-C$ satisfies C demands (users), and every schedule satisfying C users gives an integral flow of cost C. $\square$

3 Resource Transfers with Two Demands per User

If the users have more than one demand, the problem of maximizing the number of satisfied users becomes NP-hard, even APX-hard. We will show the hardness even for the special case of "using the car to commute between two badly connected locations", i.e., for the setting with two locations A and B, where every user has exactly two transfer demands, one per each direction $A \to B$, $B \to A$.

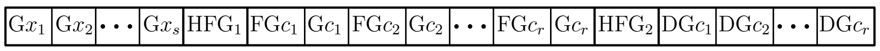


Fig. 2. Placement of the gadgets for the given instance Φ of Max-3-Sat(3) with clauses $C = \{c_1, c_2, \ldots, c_r\}$ over a set of Boolean variables $X = \{x_1, x_2, \ldots, x_s\}$.

We refer to this optimization problem as TransfersForCommuting. We actually show that the problem is APX-hard even if there is only a single resource. We prove the hardness of TransfersForCommuting by an L-reduction (see Proposition 1) from a Max-3-Sat(3), which is an APX-hard variant [1] of the maximum satisfiability problem with at most 3 literals per clause and with each variable appearing in the formula at most twice as a positive, and (exactly) once as a negative literal.

Theorem 2. TransfersForCommuting *is APX-hard even if there is only one resource, originally placed at location A, and all the transfer times are equal, but positive.*

First we describe a construction used to prove Theorem 2 and its properties.

Construction. Let Φ be an instance of Max-3-Sat(3) given by a set of clauses $C = \{c_1, c_2, \ldots, c_r\}$ over a set of Boolean variables $X = \{x_1, x_2, \ldots, x_s\}$. We construct from Φ the following instance I of TransfersForCommuting. There is a single resource in the system, initially located at A. There are two users for each occurrence of a variable in a clause and there are 26 users for each of the r clauses, in total there are at most $32r$ users. In the following we describe how the demands of the users are organized into gadgets, how they are placed, and how they interact.

For every variable $x_i \in X$ there is a *variable gadget* Gx_i. For each clause $c_j \in C$, there is a *clause gadget* Gc_j, a *dummy gadget* DGc_j, and a *light forcing gadget* FGc_j. Finally, there are two *heavy forcing gadgets* HFG$_1$ *and* HFG$_2$. The gadgets are placed as follows (see Figure 2). First all the variable gadgets are placed (one per variable in Φ). After that, the heavy forcing gadget HFG$_1$ is placed. Then, all the clause gadgets are placed (one per clause in Φ), each preceded by a light forcing gadget. Then the heavy forcing gadget HFG$_2$ is placed. Finally, all the dummy gadgets are placed (again, one per clause in Φ).

For each occurrence of a variable x_i in a clause c_j there is one variable user, demanding a transfer $B \to A$ first and $A \to B$ later, and a dummy variable user demanding a transfer $A \to B$ first and $B \to A$ later. Their outbound demands are placed in Gx_i. The return demand of the variable user is placed in Gc_j, and the return demand of the dummy variable user is placed in DGc_j. For each clause c_j, there is one clause user, demanding an outbound transfer $B \to A$, placed in Gc_j, and a return transfer $A \to B$, placed in DGc_j. Finally, there is a large number of forcing users, each demanding a transfer $A \to B$ and an immediate return $B \to A$, both placed in one of the forcing gadgets FGc_j, HFG$_1$, or HFG$_1$.

We now describe the placement of the transfers within each gadget in more detail, see Figure 3 for the exact configurations. Each light forcing gadget FGc_j,

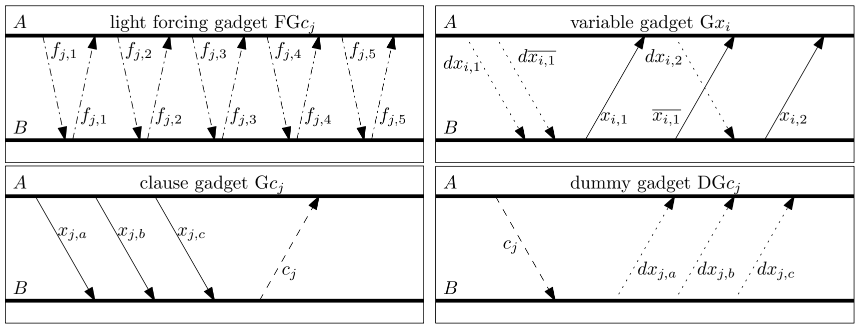


Fig. 3. Placement of the demanded transfers within the gadgets (building blocks of the hardness construction). The demands of variable users are displayed as full arrows, the demands of clause users are dashed, those of dummy variable users are dotted, and those of forcing users are dash-dotted. The heavy forcing gadget is not illustrated in the figure, since it is similar to the light forcing gadget, but consists of $10r$ users instead of 5.

consists of five light forcing users, each demanding a transfer $A \to B$ and an immediate return $B \to A$. These demands are placed in such a way that all the users of a light forcing gadget can be satisfied together. Both heavy forcing gadgets HFG$_1$ and HFG$_2$ are similar to light forcing gadgets, but instead of 5 users, each HFG consists of $10r$ heavy forcing users (again demanding transfer $A \to B$ and an immediate return $B \to A$, placed in such a way that all can be satisfied together). The purpose of the light/heavy forcing gadgets is to ensure that at a certain moment the resource is located at A. Each forcing gadget consists of a significant number of users, such that any schedule can be transformed into the same or a larger schedule, with all forcing users satisfied.

For a variable x_i, the variable gadget Gx_i consists of the outbound demands of up to six users (two for each occurrence of x_i in Φ). We describe the case when x_i appears three times in Φ, other cases are similar. The gadget Gx_i contains 3 variable users—two *positive users* $x_{i,1}$, $x_{i,2}$ corresponding to the positive literals of x_i, and one *negative user* $\overline{x_{i,1}}$ corresponding to the negative literal of x_i. Each of these users demands in Gx_i an outbound transfer $B \to A$. Additionally, the gadget contains 3 dummy variable users $dx_{i,1}$, $dx_{i,2}$, and $d\overline{x_{i,1}}$ (complementing the variable users). Again, only their outbound demands, in direction $A \to B$, are part of Gx_i. The construction of Gx_i ensures that positive and negative variable users can never be satisfied together (the demand $\overline{x_{i,1}}$ can only be fulfilled when $x_{i,1}$ and $x_{i,2}$ are not, and vice versa). Moreover, the demands of each variable user and the demands of the corresponding dummy variable user can always be fulfilled together. These gadgets relate satisfying of positive/negative variable users of I with the true/false assignment of the corresponding variables in Φ.

For a clause c_j, the clause gadget Gc_j contains the return demands of up to 3 variable users (in the direction $A \to B$) that correspond to the variables appearing in c_j, and then an outbound demand $B \to A$ of a clause user c_j.

(To simplify the notation we use c_j to denote both the clause of Φ and the corresponding user.) Each Gc_j is preceded by FGc_j enforcing that whenever the demand of c_j is fulfilled in Gc_j, also a variable demand corresponding to a literal of c_j is fulfilled there, and vice versa. Thus, these gadgets bind together clause and variable users: User c_j is satisfied if and only if the variable user of a variable satisfying c_j in Φ is satisfied.

A dummy gadget DGc_j consists of the return demand of the clause user c_j, in the direction $A \to B$, and the return demands of up to 3 dummy variable users (again, based on the literals appearing in c_j) in the direction $B \to A$. Gadget DGc_j in a sense mirrors Gc_j and allows fulfilling the return demand of c_j whenever its outbound demand is fulfilled. In a schedule where all the forcing users are satisfied, for every satisfied c_j, also a variable user and a dummy variable user (both corresponding to the same literal of c_j) will be satisfied.

Let I be an instance of TransfersForCommuting constructed as above.

Lemma 1. *Given a schedule S of I, we can construct a schedule of size at least $|S|$ where all the users of heavy forcing gadgets* HFG$_1$ *and* HFG$_2$ *are satisfied.*

Proof. If $|S| < 25r$, we can construct a schedule where all the $25r$ users of heavy and light forcing gadgets are satisfied. Thus, assume that $|S| \geq 25r$. Since the total number of users of I is at most $32r$ and each HFG consists of $10r$ users, at least one user of each heavy forcing gadget HFG is satisfied. Clearly, whenever a user of a HFG is satisfied in S, all users of that HFG can be added to S. $\square$

Lemma 2. *Given a schedule S of I, we can construct a schedule of size at least $|S|$ where all the users of light forcing gadgets are satisfied, and whenever a clause user c_j is satisfied, also a variable user corresponding to an occurrence of a literal in c_j is satisfied, as well as the corresponding dummy variable user.*

Proof. Using an iterative transformation of the given schedule S into a new one with the required parameters. The details are omitted due to space constraints.

To prove APX-hardness in Theorem 2, we use the following proposition.

Proposition 1 (L-reduction [6]). *Consider two optimization problems H and P, and let H be APX-hard. Assume that for each instance Φ of H, we can construct an instance I of P in polynomial time. Also, assume that for each solution S of I, we can construct a solution ϕ of Φ in polynomial time. Let* OPT(I), *and* OPT(Φ) *denote the size of the optimum solution of I, and Φ, respectively. Finally, assume that there exist positive constants α and β (independent on S) such that the following two conditions are met.*

(A) $\mathrm{OPT}(I) \leq \alpha\,\mathrm{OPT}(\Phi)$
(B) $|\mathrm{OPT}(\Phi) - |\phi|| \leq \beta|\mathrm{OPT}(I) - |S||$

Then, we have an L-reduction *from H to P, and P is also APX-hard.*

Proof (Of Theorem 2). We show the APX-hardness by an L-reduction from Max-3-Sat(3). Given an instance Φ of Max-3-Sat(3) with r clauses over s variables, construct an instance I of TransfersForCommuting as above. First we show the following: ($\Rightarrow$) For every solution ϕ of Φ of size $|\phi|$, we construct a solution of I of size at least $25r + 3|\phi|$. ($\Leftarrow$) For every solution S of I of size $|S|$, we construct a solution ϕ of Φ of size at least $(|S| - 25r)/3$. Thus, in particular, we get $\mathrm{OPT}(I) = 25r + 3\,\mathrm{OPT}(\Phi)$.

($\Rightarrow$) Given an assignment ϕ satisfying $|\phi|$ clauses of the given instance Φ of the Max-3-Sat(3) problem, we construct a schedule where all $25r$ forcing users together with exactly $3|\phi|$ other users are satisfied as follows. We schedule $|\phi|$ clause users corresponding to the $|\phi|$ satisfied clauses. We select a subset of (dummy) variable users: For each clause c_j, we select exactly one literal of those that satisfy c_j in ϕ and we schedule both the corresponding variable user and dummy variable user. We schedule all the users of the forcing gadgets. Let us now observe that the created schedule is feasible. Clearly, the transfers of the satisfied users do not overlap: The only overlapping transfers are those of positive/negative literals in variable gadgets and those are never satisfied together, since every variable is set either to `TRUE`, or to `FALSE` in ϕ. We now observe that the movement of the resource induced by the selected transfers is feasible. In each variable gadget the resource is moved $A \to B \to A$ for every picked literal: $A \to B$ by a dummy variable user and then $B \to A$ by the variable user. After all the variable gadgets, the resource is moved $10r$ times $A \to B \to A$ by the users of the forcing gadget HFG_1. In each clause gadget $\mathrm{G}c_j$ the resource is moved $A \to B$ by a variable user and then $B \to A$ by the clause user. Before each variable gadget, the resource is moved five times $A \to B \to A$ by the users of the forcing gadget $\mathrm{FG}c_j$. After the last clause gadget, the resource is moved $10r$ times $A \to B \to A$ by the users of HFG_2. Finally, in each dummy gadget the resource moves $A \to B \to A$.

($\Leftarrow$) Now assume that we have a schedule S with $|S|$ satisfied users. It follows from Lemma 1 and Lemma 2 that there is a schedule S' of size at least $|S|$, where all $25r$ forcing users are satisfied. Moreover, it also follows that at least $(|S| - 25r)/3$ clause users are satisfied, such that for each of them also a variable user corresponding to an occurrence of a literal in c_j is satisfied, as well as the corresponding dummy variable user are satisfied. Since the variable gadgets ensure that for each variable, either the users corresponding to the positive literals can be satisfied, or only the user corresponding to the negative literal can be satisfied, we can directly construct an assignment for Φ that satisfies at least $(|S| - 25r)/3$ clauses.

To show that the above reduction is an L-reduction, we need to prove that conditions (A) and (B) of Proposition 1 are met. First note that $\mathrm{OPT}(\Phi) \geq r/2$ (either all-`TRUE` or all-`FALSE` assignment satisfies at least 1/2 of all the clauses). Recall that $\mathrm{OPT}(I) = 25r + 3\,\mathrm{OPT}(\Phi)$. Also recall that for any solution S of I, we can construct a solution ϕ of Φ of size $|\phi| \geq (|S| - 25r)/3$. Thus we get:

(A) $\mathrm{OPT}(I) = 25r + 3\,\mathrm{OPT}(\Phi) \leq 53\,\mathrm{OPT}(\Phi)$,
(B) $(|\mathrm{OPT}(I)| - |S|) \geq 25r + 3|\mathrm{OPT}(\Phi)| - 25r - 3|\phi| = 3(|\mathrm{OPT}(\Phi)| - |\phi|)$.

It follows that the presented construction is an L-reduction from Max-3-Sat(3) to TransfersForCommuting with $\alpha = 53$ and $\beta = 1/3$. $\square$

3.1 Resource Transfers with Two Demands and Zero Transfer Times

In the hardness result, we used "crossing" arrows (demands) to exclude scheduling both demands. This is only possible if the transfer time is non-zero. In this section we show that whenever all transfer times are zero, i.e., when they are instantaneous, TransfersForCommuting becomes tractable, even if there is more than one resource (initially, a resources at location A, and b resources at location B). As a corollary, we obtain that TransfersForCommuting with non-zero transfer times is polynomially-time solvable, whenever no two demand arrows cross.

Depending on the direction of the first demanded transfer of a user, we distinguish two types of users: an ABA-type demands to transfer in direction $A \to B$ first, and in direction $B \to A$ second, whereas user of type BAB demands first the transfer in direction $B \to A$, and later in direction $A \to B$.

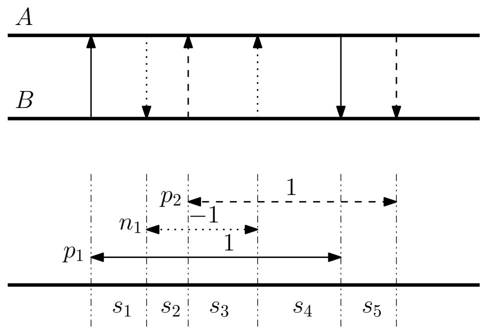
We can equivalently specify the problem with zero transfer times as follows (see an example in Figure 4). We represent the two demands of a user i (demanding instantaneous transfers at times $t_{i,1}$ and $t_{i,2}$) by a time interval $(t_{i,1}, t_{i,2})$, with a value $v(i) := -1$ if user i is of type ABA and $v(i) := 1$ if i is of type BAB. Each such time interval indicates the induced change in the number of available resources present at location A. That is, if user i of type ABA transfers a resource from A to B at time $t_{i,1}$ and back to A at time $t_{i,2}$, it implies that during the time $(t_{i,1}, t_{i,2})$ there is one less resource item at A (and one more at B).

Clearly, satisfying a user i in the original problem corresponds to selecting the corresponding interval i in the modified problem. In the original problem, there must be a resource available for each selected transfer, but since the transfers are instantaneous, we only need to ensure that, at any time, both locations have a non-negative amount of resources. In particular, at A there can never be more than $a + b$ items and less than 0 items. Therefore, the original goal translates to choosing a maximum subset I of the intervals (representing the users) such that at any time point t we have

$$-a \leq \sum_{t \in i \in I} v(i) \leq b.$$

In the following, we show that the problem (in the equivalent alternative formulation) is polynomially solvable by formulating it as an integral linear program (i.e., linear program that has an optimum solution which is integral). To prove that the constructed linear program $Ax \leq b$ is integral, we will show that A is totally unimodular (a matrix A is *totally unimodular* if the determinant of every square submatrix of A has value -1, 0 or 1).

Theorem 3 ([12]). *Every linear program in variables x with a totally unimodular constraint matrix A is integral.*



$$\begin{array}{c|ccc|}
 & p_1 & p_2 & n_1 \\
s_1 & 1 & 0 & 0 \\
 & -1 & 0 & 0 \\
s_2 & 1 & 0 & -1 \\
 & -1 & 0 & 1 \\
s_3 & 1 & 1 & -1 \\
 & -1 & -1 & 1 \\
s_4 & 1 & 1 & 0 \\
 & -1 & -1 & 0 \\
s_5 & 0 & 1 & 0 \\
 & 0 & -1 & 0 \\
p_1 & 1 & 0 & 0 \\
 & -1 & 0 & 0 \\
p_2 & 0 & 1 & 0 \\
 & 0 & -1 & 0 \\
n_1 & 0 & 0 & 1 \\
 & 0 & 0 & -1
\end{array}
\begin{pmatrix} p_1 \\ p_2 \\ n_1 \end{pmatrix}
\leq
\begin{pmatrix} b \\ a \\ b \\ a \\ b \\ a \\ b \\ a \\ b \\ a \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Fig. 4. An example of TRANSFERSFORCOMMUTING with zero transfer time, with 3 users; the transformation to an equivalent problem; and the corresponding system of linear inequalities $Ax \leq b$.

To show that A is totally unimodular, we use the following theorem [9].

Theorem 4 (Ghouila-Houri). *A matrix is totally unimodular if and only if for every subset of rows R, there exists a function $f : R \to \{-1, +1\}$ such that $\sum_{r \in R} f(r) \cdot r \in \{-1, 0, 1\}^n$.*

Theorem 5. *The problem* TRANSFERSFORCOMMUTING *with zero transfer time and multiple resources is solvable in polynomial time.*

Proof. Consider the following integer linear program formulation of the problem. Let n^+ be the number of positive intervals (i.e., intervals of value 1) and let n^- be the number of negative intervals (then $n = n^+ + n^-$). For each interval we define one variable indicating whether this interval was chosen into the optimum solution or not. In particular, for each positive interval i we define a binary variable $p_i \in \{0, 1\}$ and for each negative interval j we define a binary variable $n_j \in \{0, 1\}$. The goal is to maximize

$$\sum_{i=1}^{n^+} p_i + \sum_{j=1}^{n^-} n_j$$

subject to the following constraints. We divide the time axis into N segments, defined by the endpoints of the n given intervals. Note that the number of resources present at A may change from segment to segment, but within each segment it does not change. For each segment s we have the following two constraints, based on the number of intervals that overlap s. We abuse the notation here, and use p_i and n_j both as a variable and as the corresponding interval.

$$+ \sum_{i \in [n^+],\ p_i \cap s \neq 0} p_i \quad - \sum_{j \in [n^-],\ n_j \cap s \neq 0} n_j \leq b \qquad (1)$$

$$- \sum_{i \in [n^+],\ p_i \cap s \neq 0} p_i \quad + \sum_{j \in [n^-],\ n_j \cap s \neq 0} n_j \leq a \qquad (2)$$

We now consider the linear relaxation of the ILP, i.e., we additionally have the linear constraints, for every $i \in [n^+]$ and $j \in [n^-]$,

$$0 \leq p_i, n_j \leq 1. \tag{3}$$

We can write the constraints as a linear system $Ax \leq b$ (see Figure 4 for an example). To show that this linear program is integral, we show that the matrix A is totally unimodular. For that let us first dwell into the structure of the matrix. Matrix A contains one column for each variable (p_i or n_j) and one row for each constraint. We first discuss the first $2N$ rows corresponding to the constrains (1) and (2). For each segment s, the matrix A contains 2 rows. If the segment s coincides with an interval p_i, then the submatrix $A(s, p_i)$ is $(1, -1)^T$, otherwise, $A(s, p_i) = (0, 0)^T$. Similarly, if s coincides with an interval n_j, then $A(s, n_j) = (-1, 1)^T$, otherwise, $A(s, n_j) = (0, 0)^T$. Since all p_i and n_j are intervals, each of them spans only consecutive segments. Thus, each column (restricted to the first $2N$ rows) contains exactly one contiguous block of non-zero entries (alternating 1s and -1s). We now look at the remaining $2n = 2(n^+ + n^-)$ rows of A corresponding to the constraints (3). Clearly, each of these rows contains exactly one non-zero symbol per row and it is either 1 or -1.

Using Theorem 4, we show that A is totally unimodular as follows. We first observe that if the first $2N$ rows of A form a totally unimodular matrix A', then the whole A is totally unimodular. Each of the last $2n$ rows contains exactly one non-zero element that is either 1 or -1. Thus, for every subset R'' of these rows, we can easily find a function $f : R'' \to \{-1, +1\}$ so that each component of the vector $v'' = \sum_{r \in R''} f(r) \cdot r$ is either 0, or we can choose between 1 and -1. Then, for any vector $v' = \{-1, 0, 1\}^n$ (any vector obtained from A' due to Theorem 4) we can choose the components of $v'' = \{0, -1/1\}^n$ so that $v' + v'' = \{-1, 0, 1\}^n$. It remains to be shown that the submatrix A' corresponding to the first $2N$ rows of A is totally unimodular. Let R' be a subset of the $2N$ rows. As a preparatory step we multiply every second row of A' by -1 and obtain a matrix where each column contains a single nonzero block of either 1s (if it corresponds to p_i) or -1s (if it corresponds to n_j). We set the function $g : R' \to \{-1, +1\}$ to be alternating 1 and -1 for the $r \in R'$ ordered by row number. Since each column c contains only one block of consecutive 1s or -1s, we get $\sum_{r \in R'} g(c_r) \cdot c_r \in \{-1, 1\}$. Now we can combine the function g with the preparatory step and obtain $f : R' \to \{-1, +1\}$ such that $\sum_{r \in R'} f(r) \cdot r \in \{-1, 0, 1\}^n$.

Thus, the matrix A is totally unimodular, the constructed linear program is integral and the considered problem is polynomially solvable. □

Corollary 1. *If all the demands do not cross in their arrow representation, the problem* TransfersForCommuting *is solvable in polynomial time.*

Proof. By shrinking the given instance so that all the intervals are of length 0, we obtain an equivalent, polynomially solvable problem. □

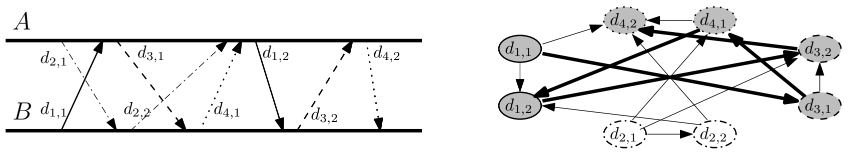


Fig. 5. An example of TRANSFERSFORCOMMUTING with 1 resource in the system, modeled as longest path problem with prescribed pairs of vertices. The edges in bold indicate the longest path.

4 Further Notes

Longest path containing subset of prescribed pairs of vertices. By proving hardness in Theorem 2, we prove also the following problem to be APX-hard. Given a directed graph and a set of pairs of its vertices, the goal is to find a longest path such that for each of the given pairs it either contain both vertices or none of them. This problem is APX-hard even in directed acyclic graphs, since it can be reduced from TRANSFERSFORCOMMUTING with 1 resource as follows (see Figure 5). Every demand is modeled as one vertex, every user defines one prescribed pair of vertices, and there is one directed edge for every pair of demands that can be consecutively fulfilled. The constructed graph is acyclic. Clearly, any path that uses from each pair either none or both vertices corresponds to a feasible schedule for TRANSFERSFORCOMMUTING and the length of the path corresponds to twice the number of satisfied users.

We haven't found this exact problem to be studied in the literature, but we link to a similar problem that received a lot of attention. Given a directed graph and a set of vertex pairs, the goal of the *longest antisymmetric path problem* is to find a longest path that does not simultaneously contain both vertices of any of the prescribed forbidden pairs. This problem arises in the area of automatic software testing and validation, and protein identification in bioinformatics. Gabow et al. [8] showed that deciding whether there is an antisymmetric st-path is NP-complete even if the given directed graph is acyclic and all the in- and out-degrees are at most 2. Song et al. [13] showed that the longest antisymmetric path problem cannot be approximated within $(n-2)/2$ in polynomial time unless P=NP, even in directed acyclic graphs of degree at most 6.

Any of multiple demands satisfies user. Consider a different variant of the problem, where each user has multiple demands, but is satisfied if any of her demands is fulfilled. It turns out, that this problem, in general form where also a transfer from a location L back to location L is allowed, is NP-hard. To see that, we consider a degenerated problem as follows. There are two locations A and B and exactly one unit of resource placed at each of them. There are n users and each user demands exactly one $A \to A$ transfer and one $B \to B$ transfer. Every transfer can take different amount of time. Clearly, we can directly model this problem as follows. We translate each demand into an interval, which is located either at A or at B. Thus, each user has exactly one interval on A and one on

B, we satisfy the user by selecting either of her intervals, and the goal is to maximize the number of satisfied users. This problem is also known under the name IntervalSelection with 2 machines, and is known to be NP-hard [4].

Open problems. We already know (Theorem 2) that the problem TransfersForCommuting where each user has exactly one demand in each of the two directions is APX-hard. Thus, a natural question is to seek an approximation algorithm for the problem. However, it is not clear whether it has a decent approximation, since simple approaches fail drastically. Also, given the motivation, it would be interesting to explore the problems we studied under online setting.

Acknowledgements. The authors wish to thank P. Widmayer for many useful discussions and helpful comments, as well as A. Bärtschi, B. Geissmann, S. Montanari, T. Pröger, and T. Tschager for their ideas during early stage discussions on the topic. Kateřina Böhmová is a recipient of the Google Europe Fellowship in Optimization Algorithms, and this research is supported in part by this Google Fellowship.

TODO!!!

References

1. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M.: Complexity and approximation: Combinatorial optimization problems and their approximability properties. Springer Science & Business Media (2012)
2. Bar-Yehuda, R., Halldórsson, M.M., Naor, J., Shachnai, H., Shapira, I.: Scheduling split intervals. SIAM Journal on Computing 36(1), 1–15 (2006)
3. Blin, G., Fertin, G., Vialette, S.: New results for the 2-interval pattern problem. In: Combinatorial Pattern Matching. pp. 311–322. Springer (2004)
4. Böhmová, K., Disser, Y., Mihalák, M., Widmayer, P.: Interval selection with machine-dependent intervals. In: WADS 2013. pp. 170–181 (2013)
5. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to algorithms, vol. 3. MIT Press Cambridge, MA (2001)
6. Crescenzi, P.: A short guide to approximation preserving reductions. In: 12th IEEE Conference on Computational Complexity. pp. 262–273. IEEE (1997)
7. Crochemore, M., Hermelin, D., Landau, G.M., Vialette, S.: Approximating the 2-interval pattern problem. In: ESA 2005, pp. 426–437. Springer (2005)
8. Gabow, H.N., Maheshwari, S.N., Osterweil, L.J.: On two problems in the generation of program test paths. Software Eng., IEEE Transactions on 2(3), 227–231 (1976)
9. Ghouila-Houri, A.: Caracterisation des matrices totalement unimodulaires. CR Acad. Sci. Paris 254, 1192–1194 (1962)
10. Kolen, A.W.J., Lenstra, J.K., Papadimitriou, C.H., Spieksma, F.C.R.: Interval scheduling: a survey. Naval Research Logistics (NRL) 54(5), 530–543 (2007)
11. Kovalyov, M.Y., Ng, C., Cheng, T.E.: Fixed interval scheduling: Models, applications, computational complexity and algorithms. European Journal of Operational Research 178(2), 331–342 (2007)
12. Schrijver, A.: Theory of linear and integer programming. John Wiley & Sons (1998)
13. Song, Y., Yu, M.: On finding the longest antisymmetric path in directed acyclic graphs. Information Processing Letters 115(2), 377–381 (2015)