SIGACT News Online Algorithms Column 31

Rob van Stee
University of Siegen
Siegen, Germany

Today I am very happy to present a full version of the proof that the classic List Update problem is NP-hard, by Christoph Ambühl. This proof first appeared in ESA 2000. It was later submitted to a journal, but the refereeing process got stalled as Christoph left academia. Fortunately he has now agreed to have his complete and final version appear in this column.

What makes this particularly pleasing is that the proof is now significantly easier than in the ESA version: there is just one type of gadget left, and this gadget itself has been slightly changed, leading to a much easier analysis. Another change from the conference version is that the final reduction of the weighted to the unweighted problem is now done using the probabilistic method.

I am very pleased to be able to fill this gap in the literature with this column.

As always, I would like to invite more contributions to this column, be it surveys, conference reports, or technical articles related to online algorithms and competitive analysis. If you are considering becoming a guest writer, don't hesitate to mail me at `rob.vanstee@uni-siegen.de`.

Offline List Update is NP-hard

Christoph Ambühl*

Abstract

The list update problem is a well studied online problem in the area of self-adjusting data structures. Understanding the offline version of this problem is crucial because of the role it plays in the competitive analysis of online list update algorithms. In this paper we settle a long-standing open problem by showing that the offline list update problem is *NP*-hard.

1 Introduction

The *list update problem* is a classical online problem in the area of self-organizing data structures [3, 6]. It is concerned with unsorted linear lists as an implementation of a dictionary data structure.

Requests to items must be served by accessing the requested item in the list. Accessing the item at position i in the list incurs a cost of i units. The goal is to keep access costs small by rearranging the items in the list. After an item has been requested, it may be moved free of charge closer to the front of the list. This is called a *free exchange*. The other way of rearranging the list items is by swapping two adjacent items, which is called a *paid exchange*. Paid exchanges can be performed at any time. But each of then incurs a cost of one unit.

A list update algorithm describes the policy by which the items are rearranged. The most famous of all is the MoveToFront algorithm (MTF), which moves each item to the front of the list after it has been requested.

An *online* algorithm must serve the sequence σ of requests one item at a time, without knowledge of future requests. An optimum *offline* algorithm knows the entire sequence σ in advance and can serve it with minimum cost $OPT(\sigma)$. If the online algorithm serves σ with cost $A(\sigma)$, then it is called c-*competitive* if for a suitable constant b

$$A(\sigma) \leq c \cdot OPT(\sigma) + b \tag{1}$$

for all request sequences σ and all initial list states. The constant c is called the *competitive ratio* of A.

In a seminal paper, Sleator and Tarjan proved that MTF is 2-competitive [13]. This is best possible for deterministic algorithms (see the remark in [12]).

As shown first by Irani [10], *randomized* algorithms can perform better on average. A randomized algorithm is called c-competitive if

$$E[A(\sigma)] \leq c \cdot OPT(\sigma) + b,$$

for all σ and all initial list states, where the expectation is taken over the randomized choices of the online algorithm. The best randomized list update algorithm known to date is the 1.6-competitive algorithm COMB by Albers,von Stengel and Werchner [2]. COMB is a combination

*The research resulting in this paper was conducted while the author was a Ph.D. student at the Institute for Theoretical Computer Science, ETH Zürich, Switzerland.

of the BIT algorithm by Reingold, Westbrook, and Sleator [12] and Albers' TIMESTAMP algorithm [1]. The former is chosen with probability 4/5, the latter with probability 1/5.

For the analysis of list update algorithms, the model in which accessing the item at position i costs only $i-1$ units is more natural. This model is called *partial cost model*, as opposed to the classical *full cost model*. An algorithm which is c-competitive in the partial cost model is also c-competitive in the full cost model. On the other hand, a lower bound in the full cost model implies the same bound in the partial cost model, but not vice versa.

The first nontrivial lower bound for the full cost model is due to Karp and Raghavan (see the remark in [12]). The best lower bound known to date is 1.5 by Teia [14].

In the partial cost model, a lower bound of 1.5 is trivial. But the best known lower bound by Ambühl, Gärtner, and von Stengel is only marginally higher at 1.50084 [5].

The analysis of all algorithms mentioned above relies on the fact that the algorithms have the so-called projectivity property. That is, the relative order of two items x and y does not depend on items different from x and y. Ambühl, Gärtner, and von Stengel showed that in the partial cost model no projective algorithm can outperform COMB [4]. This means that new kinds of algorithms and new ways of analyzing algorithms are needed to close the gap.

Because of (1), the analysis of an online algorithm requires a good understanding of the optimum offline algorithm OPT. Some online problems, for example Paging [6], benefit from the fact that they have a polynomial time optimum offline algorithm. These algorithms tend to be fairly simple and therefore help a lot in finding good online algorithms.

Unfortunately, this is probably not the case for the list update problem, as the main result of this paper is

Theorem 1. *The Offline List Update Problem (OLUP) is NP-hard.*

This theorem holds in both cost models. Reingold and Westbrook provided the best published algorithm for OLUP [11]. It runs in time $O(2^n n!|\sigma|)$ on lists with n items. More recently Divakaran claimed an improved running time of $O(nn!|\sigma|)$ [8].

2 Preliminaries

An instance of the *Offline List Update Problem* (OLUP) consists of a list state L_0 and a sequence of requests σ. The objective is to determine the cheapest way to serve all the requests in σ in turn starting from the initial list L_0. We denote the minimal cost needed to serve an instance (L_0, σ) by $OPT(L_0, \sigma)$.

Throughout this paper, we assume the partial cost model. However, the cost of a schedule $\mathcal{S}$ in the full cost model can be obtained by adding $|\sigma|$ units to the cost of $\mathcal{S}$ in the partial cost model. Therefore, OLUP is certainly *NP*-hard in the full cost model as well.

List states will be written in brackets, with the head of the list on the left. A request sequence is denoted by a sequence of items from L_0. For example $L_0 = [abc]$ and $\sigma = cbbc$. In order to denote that x is before y in a list state, we write $x \prec y$. The ith request of σ is denoted by $\sigma(i)$. By $\sigma'\sigma''$ we denote the concatenation of two sequences σ' and σ''. We denote the request sequence consisting of $q \geq 0$ repetitions of σ by σ^q.

The number of requests in σ will be denoted by $|\sigma|$. By σ_{xy} we denote the *projection of σ to the items x and y*. That is, the sequence obtained from σ after all requests to items other than x and y have been removed. The length of σ_{xy} is denoted by $|\sigma_{xy}|$. In this vein, the number of requests to item x in σ will be denoted by $|\sigma_x|$.

Theorem 3 in [13] states that paid exchanges can be mimicked by free exchanges. Reingold and Westbrook [11] give a counter-example to this claim: Consider L_0 and σ from two paragraphs above. An optimal algorithm moves a behind b and c before the first request to c. This requires paid exchanges and incurs eight cost units. All solutions using only free exchanges incur at least cost nine.

On the other hand, free exchanges can be mimicked by paid exchanges as follows: Instead of first paying k units in order to access item x and then move it at no charge t positions closer to the front, one can first move the item t positions and then access the item. In both cases, one pays exactly k units.

Because we can ignore free exchanges, a solution to an instance of OLUP is determined by the sequence of list states $\mathcal{S} = (L_0, L_1, \ldots, L_{|\sigma|})$ where L_i for $1 \leq i \leq |\sigma|$ denotes the state in which the ith request is performed, and L_0 denotes the initial list state. A feasible (but not necessarily optimal) solution $\mathcal{S}$ will be called a *schedule*.

Concerning the cost of a schedule $\mathcal{S}$, the access cost for the ith request can easily be determined from L_i. Namely, if the requested item is at position p in L_i, then accessing it costs $p-1$ units. (Remember that we decided to use the partial cost model.)

The cost of the paid exchanges needed to turn a list state L_i into L_{i+1} is equal to the number of *inversions* between L_i and L_{i+1}. That is, the number of pairs of items whose relative order in L_i is reversed in L_{i+1}.

We will show that computing $OPT(L_0, \sigma)$ is *NP*-hard by showing that the decision version of the problem is *NP*-hard. Therefore, there cannot be a polynomial time algorithm for OLUP unless $P = NP$.

In the next section, we will introduce a generalized version of OLUP, called *Weighted List Update Problem* (WLUP). In Section 4, we will reduce the well known Minimum Feedback Arc Set Problem to WLUP to show that WLUP in NP-hard. In the Section 5, we will finally show that OLUP is NP-hard by a reduction from WLUP.

3 The Weighted List Update Problem

In this section, we introduce the Weighted List Update Problem (WLUP), which generalizes OLUP to items with *weights*. These weights have to be non-negative integers. Weighted items have been considered already in [7] for online list update.

The cost incurred by operating on weighted items is the following. The cost of a paid exchange involving two adjacent items x_i and x_j with weights w_i and w_j is

$$w_i \cdot w_j \tag{2}$$

units. The access cost for item x_i in a list L is

$$\sum_{k:x_k \prec x_i \text{ in } L} w_k \cdot w_i. \tag{3}$$

An instance of WLUP consists of a request sequence σ and an initial list L_0 over a set of weighted items. We denote an instance by the triple (L_0, σ, W), where W is a vector that contains the weight of each item in L_0. We denote the minimal cost to serve an instance by $WOPT(L_0, \sigma, W)$.

Note that the special case where all items have unit weight is equivalent to the classical list update problem.

For the rest of this section, we consider a fixed WLUP instance (L_0, σ, W). The cost of a schedule $\mathcal{S}$ for (L_0, σ, W) can be described by the two functions *upd* and *jmp*, which both map to $\{0, 1\}$. Let $upd(i, \{x, y\}, \mathcal{S}) = 1$ if and only if the relative order of x and y changes from L_{i-1} and L_i. The cost for changing the list state from L_{i-1} to L_i can then be written as

$$\sum_{\{x,y\}} upd(i, \{x, y\}, \mathcal{S}) w_x w_y.$$

What the above formula shows is that every unit of cost incurred by updating the list from L_{i-1} to L_i can be assigned to exactly one unordered pair of items. The same can be done for

the access cost. When serving $\sigma(i)$, a pair $\{x, y\}$ incurs a cost if either x is accessed and y is in front of x in L_i, or if y is accessed and x is in front of y. In this vein, let $jmp(i, \{x, y\}, \mathcal{S}) = 1$ if and only if

$$(\sigma(i) = x \text{ and } y \prec x \text{ in } L_i) \text{ or } (\sigma(i) = y \text{ and } x \prec y \text{ in } L_i).$$

The access cost for the ith request in the request sequence can then be written as

$$\sum_{\{x,y\}} jmp(i, \{x, y\}, \mathcal{S}) w_x w_y.$$

Using *jmp* and *upd*, the cost of a schedule $\mathcal{S}$ can be expressed as

$$\Gamma(\mathcal{S}) = \sum_{1 \le i \le |\sigma|} \sum_{\{x,y\}} w_x w_y \big(upd(i, \{x, y\}, \mathcal{S}) + jmp(i, \{x, y\}, \mathcal{S}) \big). \tag{4}$$

The *projected cost of the pair* $\{x, y\}$ in a schedule $\mathcal{S}$ for σ is defined by

$$\gamma_{xy}(\sigma, \mathcal{S}) := \sum_{1 \le i \le |\sigma|} upd(i, \{x, y\}, \mathcal{S}) + jmp(i, \{x, y\}, \mathcal{S}).$$

Note that γ_{xy} does not depend on W. We can now write Equation (4) as

$$\Gamma(\mathcal{S}) := \sum_{\{x,y\}} w_x w_y \cdot \gamma_{xy}(\sigma, \mathcal{S}).$$

We will often need to argue about the projected cost of different parts of a request sequence. Let $\sigma(a) \dots \sigma(b)$ be a consecutive subsequence of σ. The *projected cost of the pair* $\{x, y\}$ *on the subsequence* $\sigma(a) \dots \sigma(b)$ in a schedule $\mathcal{S}$ is defined by

$$\gamma_{xy}(\sigma, a, b, \mathcal{S}) := \sum_{a \le i \le b} upd(i, \{x, y\}, \mathcal{S}) + jmp(i, \{x, y\}, \mathcal{S}).$$

We also need a simple lower bound on $\gamma_{xy}(\sigma, a, b, \mathcal{S})$. Assume that we can partition $\sigma(a) \dots \sigma(b)$ into k consecutive subsequences, with each subsequence containing at least one request to each of x and y. Clearly, each of these subsequences will incur at least one unit towards $\gamma_{xy}(\sigma, a, b, \mathcal{S})$. Therefore k is a lower bound on $\gamma_{xy}(\sigma, a, b, \mathcal{S})$. The partitioning which yields the lower bound can easily be computed in a greedy manner from the beginning to the end of $\sigma(a) \dots \sigma(b)$. Let $\delta_{xy}(\sigma(a) \dots \sigma(b))$ be the number of subsequences we can obtain by partitioning σ in the greedy manner. Then we can state that

Proposition 2. $\gamma_{xy}(\sigma, a, b, \mathcal{S}) \ge \delta_{xy}(\sigma(a) \dots \sigma(b))$.

The following properties of δ_{xy} will be useful.

Proposition 3.

$$\begin{aligned} &\delta_{xy}(\sigma) = \delta_{xy}(\sigma_{xy}), \\ &\delta_{xy}(\sigma) \ge \delta_{xy}(\sigma') \qquad \text{if } \sigma' \text{ is a subsequence of } \sigma \end{aligned}$$

If additional information about $\mathcal{S}$ is known, one can often get a better lower bound on $\gamma_{xy}(\sigma, a, b, \mathcal{S})$.

Lemma 4. *Let* $1 \le a \le q \le b \le |\sigma|$ *and let* $\mathcal{S} = (L_0, L_1, \dots, L_{|\sigma|})$ *be a schedule for* σ *with* $x \prec y$ *in* L_q. *Then*

$$\gamma_{xy}(\sigma, a, b, \mathcal{S}) \ge \delta_{xy}(\sigma(a) \dots \sigma(q-1)\, x\, \sigma(q) \dots \sigma(b)) \quad \text{and} \tag{5}$$

$$\gamma_{xy}(\sigma, a, b, \mathcal{S}) \ge \delta_{xy}(\sigma(a) \dots \sigma(q)\, x\, \sigma(q+1) \dots \sigma(b)). \tag{6}$$

Proof. We only prove Inequality (5) here. The proof of Inequality (6) is very similar. Let σ' be the request sequence obtained from σ by adding a request to x between $\sigma(q-1)$ and $\sigma(q)$. Notice that $\sigma(b)$ corresponds to $\sigma'(b+1)$ since one request to x was added to the sequence. From a schedule $\mathcal{S}$ described above one can easily build a schedule $\mathcal{S}'$ for σ' such that $\gamma_{xy}(\sigma, a, b, \mathcal{S}) = \gamma_{xy}(\sigma', a, b+1, \mathcal{S}')$ holds. Then the lemma follows from $\gamma_{xy}(\sigma', a, b+1, \mathcal{S}') \geq \delta_{xy}(\sigma'(a) \dots \sigma'(b+1))$ (Proposition 2). □

Obviously, if the relative ordering of x and y is known at two list states L_q and L_p, the technique of adding a request can be applied at both $\sigma(q)$ and $\sigma(p)$, thus leading to an even better lower bound.

The request sequence σ that we will encounter in the next section will be the concatenation of so-called gadgets. Let us introduce some special notation to handle these gadgets more easily. Let π be a gadget starting at the ath request and ending at the bth request of σ.

$$\begin{aligned} \gamma_{xy}(\pi, \mathcal{S}) &:= \gamma_{xy}(\sigma, a, b, \mathcal{S}), \\ \Gamma(\pi, \mathcal{S}) &:= \sum_{\{x,y\}} w_x w_y \cdot \gamma_{xy}(\pi, \mathcal{S}), \\ \Delta(\pi) &:= \sum_{\{x,y\}} w_x w_y \cdot \delta_{xy}(\pi) \end{aligned}$$

$\Gamma(\pi, \mathcal{S})$ is nothing else but the total cost incurred during serving the gadget π using schedule $\mathcal{S}$. It is easy to see that by summing up $\Gamma(\pi, \mathcal{S})$ for all gadgets π we obtain $\Gamma(\mathcal{S})$.

Note that for computing $\gamma_{xy}(\pi, \mathcal{S})$ and $\Gamma(\pi, \mathcal{S})$ we need to know at which request π starts and ends. On the other hand, for $\delta_{xy}(\pi)$ all we need is the request sequence of the gadget. This is also why we do not need a special definition for $\delta_{xy}(\pi)$.

These new notations allow us to rewrite and extend Proposition 2. The first inequality is just Proposition 2 using the new notation. The second inequality follows easily from the first and the definitions of $\Gamma(\pi, \mathcal{S})$ and $\Delta(\pi)$.

Proposition 5.

$$\begin{aligned} \gamma_{xy}(\pi, \mathcal{S}) &\geq \delta_{xy}(\pi), \\ \Gamma(\pi, \mathcal{S}) &\geq \Delta(\pi) \end{aligned}$$

4 WLUP is NP-hard

To show that WLUP is NP-hard, we will give a polynomial time reduction from *Minimum Feedback Arc Set Problem* (MINFAS) [9] to WLUP. Given a directed graph $G = (V, E)$, a feedback arc set is a set $E' \subseteq E$ such that $G' = (V, E \setminus E')$ is acyclic. In the decision version, we want to decide whether there exists a feedback arc set of cardinality at most k for an instance G. We can assume $k < |V|^2$ here since the problem is trivial for larger values of k.

The reason we reduce from MINFAS is that this problem can be reformulated as an ordering problem. Instead of expressing a feedback arc set by a subset of E, it can be encoded in a permutation P of the vertex set V. The permutation P in turn can be interpreted as a list state. The feedback arc set encoded in P is

$$\kappa(P) := \{(v_i, v_j) \in E \mid v_j \prec v_i \text{ in } P\}.$$

To see that $\kappa(P)$ is indeed a feedback arc set, note that any cycle in G must contain at least one arc for which $v_j \prec v_i$ holds in P. Therefore, $E \setminus \kappa(P)$ is cycle-free and $\kappa(P)$ is indeed a feedback arc set.

We also have to prove that if a graph G has a feedback arc set E' of cardinality k, then there exists a permutation P with $|\kappa(P)| \leq k$. To see this, remember that $G' = (V, E \setminus E')$ is acyclic. Therefore one can always find a permutation P of V such that for every arc $(v_i, v_j) \in E \setminus E'$ it holds that $v_i \prec v_j \in P$. Such an ordering is called a topological ordering of G'. The only arcs for which $v_j \prec v_i$ might hold in P are those in E'. Therefore, $|\kappa(P)| \leq k$ holds.

The reduction from MINFAS to WLUP is a polynomial time computable function which takes a MINFAS instance (G, k) as arguments and returns a WLUP instance (L_0, σ, W, k') such that (L_0, σ, W) has a schedule with cost k' if and only if G has a feedback arc set k.

We are now going to describe what (L_0, σ, W) looks like. Let $G = (V, E)$ and $n = |V|$. For every vertex $v_i \in V$ there is a weighted item v_i with weight $k + 1$. We call them *vertex items*. Additionally, we have two items c and d both with weight one. They will be used to implement the so-called gadgets.

The initial list state is $L_0 := [v_1 v_2 v_3 \dots v_n c\, d]$. The request sequence σ is the concatenation of subsequences called *gadgets*. There is exactly one gadget for every arc of G. The gadget for an arc $e = (v_i, v_j)$ will be denoted by $\pi(v_i, v_j)$ or $\pi(e)$.

We now have enough information to explain the high-level ideas behind the reduction. Let (G, k) be a MINFAS instance and let P be a permutation with $|\kappa(P)| \leq k$. The WLUP instance (L_0, σ, W) has a very simple schedule with value at most k'. Namely, the schedule will rearrange the vertex items within the first n requests of σ in such a way that their ordering corresponds to the ordering of the vertices in P. In the remainder of the schedule, the relative order of the vertex items will not change anymore. The purpose of each gadget $\pi(v_i, v_j)$ is to test whether $v_j \prec v_i$ holds in the list. If this is the case, serving the gadget will cost one unit more compared to the case in which $v_i \prec v_j$ holds. Clearly, there will be exactly $\kappa(P)$ gadgets which will incur this one unit of extra cost. On the other hand, we will show that from a schedule with cost k' for (L_0, σ, W) one can very easily read off a permutation P with $|\kappa(P)| \leq k$.

The details of the gadgets are as follows. Every gadget consists of eight copies of the sequence $(v_1 \dots v_n)$ with the addition of some requests to c and d. The gadget $\pi(v_i, v_j)$ can be written as follows.

$$(v_1 \dots v_n)^2\, v_1 \dots c\, v_i \dots v_j\, d \dots v_i\, cc \dots ddd\, v_j \dots v_i\, ccc \dots v_n\, (v_1 \dots v_n)^3$$

The first request to c in the gadget is always just before the third request to v_i. The first request to d follows just after the next request to v_j. Then there is a double request to c just after the next request to v_i, followed by a triple request to d just before the next request to v_j, finally, there is a triple request to c just after the next request to v_i. Note that the triple request to c always happens in the fourth copy of $(v_1 \dots v_n)$. In which copies of $(v_1 \dots v_n)$ the requests to d happen depends on whether $i > j$ or $i < j$.

Here is another view of the gadgets, where we have distinguished between the cases $i < j$ (left) and $i > j$ (right) and highlighted the important part.

$$\begin{array}{l} (v_1 \dots\ v_i\ \dots\ v_j\ \dots\ v_n)^2 \\ v_1 \dots\ \boldsymbol{c v_i}\ \dots\ \boldsymbol{v_j d}\ \dots\ v_n \\ v_1 \dots\ \boldsymbol{v_i cc}\ \dots\ \boldsymbol{ddd v_j}\ \dots\ v_n \\ v_1 \dots\ \boldsymbol{v_i ccc}\ \dots\ v_j\ \dots\ v_n \\ (v_1 \dots\ v_i\ \dots\ v_j\ \dots\ v_n)^3 \end{array} \qquad \begin{array}{l} (v_1 \dots\ v_j\ \dots\ v_i\ \dots\ v_n)^2 \\ v_1 \dots\ v_j\ \dots\ \boldsymbol{c v_i}\ \dots\ v_n \\ v_1 \dots\ \boldsymbol{v_j d}\ \dots\ \boldsymbol{v_i cc}\ \dots\ v_n \\ v_1 \dots\ \boldsymbol{ddd v_j}\ \dots\ \boldsymbol{v_i ccc}\ \dots\ v_n \\ (v_1 \dots\ v_j\ \dots\ v_i\ \dots\ v_n)^3 \end{array}$$

The crucial part of the gadget is the part between the first and the last request to c. The only facts to remember about the rest of the gadget is that every vertex item gets requested at least twice before the first request to c and at least three times after the last request to c.

Although the gadgets might look very complicated when looking at them as a whole, the inner workings are not too difficult to understand once one considers the projections to the pairs, as we will do in the next subsection.

We partition the gadgets into two sets. An arc (v_i, v_j) and its gadget $\pi(v_i, v_j)$ belongs to E^- if $i < j$, and to E^+ if $i > j$.

As already stated earlier, the request sequence σ is just the concatenation of all gadgets. They are ordered in such a way that the gadgets for arcs in E^+ precede all the gadgets for arcs in E^-. Clearly

$$\Delta(\sigma) := \sum_{e \in E} \Delta(\pi(e))$$

is a lower bound on $WOPT(L_0, \sigma, W)$. We can now complete the description of the reduction by setting $k' := \Delta(\sigma) + k$. A schedule with cost at most $\Delta(\sigma) + k$ will be called a *good schedule*.

If in a schedule $\mathcal{S}$ it holds that $\gamma_{xy}(\pi, \mathcal{S}) = \delta_{xy}(\pi)$ we say that the pair $\{x, y\}$ is *tight* in the gadget π, otherwise it incurs $w_x w_y \cdot [\gamma_{xy}(\pi, \mathcal{S}) - \delta_{xy}(\pi)]$ units of *extra cost*. If in a gadget π all pairs are tight and therefore $\Gamma(\pi, \mathcal{S}) = \Delta(\pi)$, the gadget π is called *tight* as well.

The idea behind the gadgets is the following. Consider a gadget $\pi(v_i, v_j)$ and let $L^\#$ be the list state in which the gadget's first request to d takes place. The purpose of the gadget is to test whether $v_i \prec v_j$ holds in $L^\#$. In that case, the cost of serving the gadget will be equal to $\Delta(\pi)$, otherwise the cost will be $\Delta(\pi) + 1$.

To show that the reduction works we prove two claims. The first one is

Claim 6. *If G has a permutation P with $|\kappa(P)| \leq k$, then there is a schedule for (L_0, σ, W) with cost $\Delta(\sigma) + |\kappa(P)| \leq \Delta(\sigma) + k$.*

Since we have a gadget for every arc in G, all we have to do is to rearrange the vertex items such that their order is equivalent to P. As we will see, this reordering does not incur any extra cost. With the vertex items ordered according to P, all the gadgets $\pi(v_i, v_j)$ with $v_j \prec v_i$ in P will incur one additional cost unit. All other gadgets will not incur any extra cost. Hence the cost will be bounded by $\Delta(\sigma) + |\kappa(P)| \leq \Delta(\sigma) + k$. The second claim is

Claim 7. *If there is a schedule $\mathcal{S}$ for (L_0, σ, W) with cost at most $\Delta(\sigma) + k$, then G has a permutation P with $|\kappa(P)| \leq k$.*

The proof of the claim is along the following lines. Clearly there can be at most k gadgets with are not served tightly in $\mathcal{S}$. Let L^* be the list state just after the last gadget belonging to E^+ has been served in the schedule $\mathcal{S}$. The goal is to show that the permutation P, which is obtained from L^* by removing c and d, represents a feedback arc set of cardinality at most k for G.

To see this, we will prove that for all tight gadgets $\pi(v_i, v_j)$ it holds that $v_i \prec v_j$ in L^* and therefore also in P. Hence, only the at most k arcs corresponding to non-tight gadgets can have $v_j \prec v_i$ in P and therefore $|\kappa(P)| \leq k$ holds.

4.1 Proof of Claim 6

It is now time to analyze σ as a whole and particulary the gadgets in more detail. We are mostly interested in good schedules. In a good schedule, all the pairs except $\{c, d\}$ need to be served tightly in all the gadgets. This holds because the projected cost of a pair of items is always a multiple of the product of items weights, hence serving a pair of items other than $\{c, d\}$ non tightly would incur extra cost of at least $k + 1$ units. It turns out that there is only limited amount of freedom when it comes to serving a gadget tightly.

Lemma 8. *Let $\pi = \pi(v_i, v_j)$. We have $\delta_{v_\ell c}(\pi) = 4$ for all $\ell \in \{1, \ldots, n\}$, $\delta_{v_j d}(\pi) = 2$, and $\delta_{v_\ell d}(\pi) = 3$ for $v_\ell \neq v_j$. Finally, $\delta_{cd}(\pi) = 3$. In a good schedule, c and d are behind all vertex items at the beginning and end of all gadgets. If the pair $\{c, d\}$ is served tightly in a gadget, then we must have $c \prec d$ at the beginning and end of this gadget.*

Proof. We repeatedly use Proposition 3. In the case $\{v_i, c\}$, we have $\delta_{v_i c}(v_i^2\, cv_i\, v_i cc\, v_i ccc\, v_i^3) = 4$. This lower bound can be achieved only in one way, namely by having c behind v_i at the start of the gadget (i.e., without paying for an exchange), keeping c behind v_i until the second request to c, and by moving v_i in front of c again after the last request to c in the gadget. It is easy to check that deviating from this approach, for example by moving c in front of v_i earlier, would add extra cost. It is not hard to see, using Lemma 4, that any other schedule would incur extra cost.

In the case of $\{v_\ell, c\}$ for $v_\ell \neq v_i$, we have $\delta_{v_\ell c}(v_\ell^p cv_\ell ccv_\ell cccv_\ell^{6-p}) = 4$ for $p \in \{2, 3\}$ and there are exactly two ways to serve this pair. At the start, c must be behind v_ℓ. Because there is only one request to v_ℓ between the first and second request to c, there is the option of moving c in front of v_ℓ either at the first or at the second request. After the last request to c in the gadget, v_ℓ has to move in front of c again.

For $\{v_j, d\}$ we have $\delta_{v_j d}(v_j^p d\, ddd v_j^{8-p}) = 2$ for $p \in \{3, 4\}$ and there is only one way to achieve this bound, namely by having d behind v_j at the start, moving d in front of v_j at the first request to d and moving it back straight after the last request to d.

For $\{v_\ell, d\}$ for $v_\ell \neq v_j$, the lower bound is $\delta_{v_\ell d}(v_\ell^p\, dv_\ell\, ddd v_\ell^{7-p}) = 3$ for $p \in \{2, 3, 4\}$. And there is again the option of moving d in front of v_ℓ either at the first or at the second request. Again, v_ℓ has to move ahead of d after the last request to d.

Finally, the pair $\{c, d\}$ has $\delta_{cd}(cdccdddccc) = 3$. The only optimal way to serve this pair is to serve all requests to c with $c \prec d$ and serve the triple request to d with $d \prec c$. In particular, in order to have projected cost 3, by Lemma 4 we must have $c \prec d$ at the beginning and end of the gadget.

By combining all of these cases, we see that c and d have to be behind *all* vertex items at the end of each gadget, and in this order ($c \prec d$) if they are served tightly. □

Using Lemma 8, we can now prove the claim. Let $G = (V, E)$ be a graph with permutation P such that $|\kappa(P)| \leq k$. We have to show that there exists a schedule $\mathcal{S}$ for (L_0, σ, W) with cost $\Delta(\sigma) + |\kappa(P)|$. This means that $\mathcal{S}$ must be a good schedule.

We define the schedule $\mathcal{S}$ as follows. We use the first n requests of the first gadget of σ to rearrange the vertex items according to the permutation P. From then on, the relative order of the vertex items will remain unchanged. To describe how this is achieved, it suffices to describe what the list states $L_1, \ldots, L_n$ of the schedule $\mathcal{S}$ look like. Namely, L_i has items $v_1, v_2 \ldots, v_i$ ordered according to P, followed by the remaining items ordered as in L_0. Hence in L_n, all the vertex items are ordered according to P, and this rearrangement of the items does not incur more cost than accessing them.

It remains to describe how c and d behave in the schedule. The items c and d are always at the tail of the list at the start of a gadget, which $c \prec d$. Just before the first request to c in the gadget, we move c right behind v_i. Then we move d right before v_j before the first request to d. Before the double request to c, we move c to the front of the list. Before the triple request to d, we move d to the front of the list. The same happens to c before its triple request. After the last request to c in the gadget, we move both c and d back to the tail of the list.

To analyze the amount of extra cost incurred in the schedule, we look at the projected cost of all pairs of items. Remember that in order to compute the projected cost of a pair in a part of a request sequence, all one needs to know is their relative order at each request to one of them (for the access cost) and how many times the pair changes its relative order (for the update cost).

Let us first check that all pairs of vertex items $\{v_x, v_y\}$ do not induce extra cost. W.l.o.g. we can assume that $x < y$. In the schedule described above, v_y either stays behind v_x for the whole schedule, or v_y passes v_x just before v_y's first request in σ, using a single paid exchange. Hence $\gamma_{v_x v_y}(\pi, \mathcal{S}) = 8$ holds in both cases for all gadgets, which is tight.

The behavior of the pairs $\{v_i, c\}$ and $\{v_j, d\}$ is completely determined by the description of the schedule: c stays behind v_i until the second request to c, whereas d moves ahead to v_j right at the first request. Both c and d move behind the other item after their last request. For the other pairs, it can easily be checked that this schedule serves them as in the proof of Lemma 8, hence serving them tightly.

Finally, the pair $\{c, d\}$ needs to be analyzed. From the description of the schedule we know that $c \prec d$ holds at the start of the gadget. If $(v_i \prec v_j)$ in P and therefore v_i is before v_j in the list during the service of the gadget (or at least after the first n requests of this gadget have been served, in case this is the first gadget), d will stay behind c at its first request, and therefore the projected cost will be three. But if $v_j \prec v_i$ holds in P, then v_j is before v_i in the list when the requests to c and d are served. In that case d passes c at the first request to d in the gadget, thus causing one extra cost unit as c passes d again at the second request to c.

Since the one unit of extra cost occurs only for gadgets $\pi(v_i, v_j)$ with $v_j \prec v_i$ in P and the number of such gadgets is $\kappa(P)$, the schedule as described incurs $|\kappa(P) \leq k$ units of extra cost. This completes the proof of Claim 6.

4.2 Proof of Claim 7

We need to prove that a good schedule $\mathcal{S}$ for (L_0, σ, W) implies that there exists a permutation P with $|\kappa(P)| \leq k$.

Lemma 9. *In a good schedule $\mathcal{S}$, the relative order of a pair of vertex items can change at most once.*

Proof. Consider a good schedule $\mathcal{S}$ and two vertex items v_x and v_y. Without loss of generality we can assume $x < y$. Clearly it holds that $v_x \prec v_y$ in L_0.

The proof is by contradiction. Assuming that the relative order of v_x and v_y changes at least twice, one can partition σ into $\sigma'\sigma''\sigma'''$ such that $v_y \prec v_x$ holds when the first request of σ'' takes place and $v_x \prec v_y$ when the first request of σ''' takes place. Applying Lemma 4 at the first request of σ'' and at the first request of σ''', one can conclude that there exist integers $p, q \geq 0$ such that $\gamma_{v_x v_y}(\sigma, \mathcal{S})$ is lower bounded by

$$\begin{aligned}
\delta_{v_x v_y}(\sigma' v_y \sigma'' v_x \sigma''') &= \delta_{v_x v_y}(\sigma'_{v_x v_y} v_y \sigma''_{v_x v_y} v_x \sigma'''_{v_x v_y}) \\
&= \delta_{v_x v_y}\left((v_x v_y)^q (v_y v_x)^{p+1} (v_x v_y)^{8|E|-q-p}\right) \\
&= 8|E| + 1.
\end{aligned}$$

The first equality follows from Proposition 3. For the second equality, it helps to notice that the sequence on the left hand side always has exactly one double request to each v_y and v_x, with the double request to v_y preceding the one to v_x. By cutting the sequence between the two double requests, one ends up with the three terms stated on the right hand side. (In case $\sigma'_{v_x v_y} = \emptyset$, v_y is inserted at the start, and we have $q = 0$. We have $\sigma'''_{v_x v_y} \neq \emptyset$ since there is no state change after the last request.)

This implies that the pair $\{v_x, v_y\}$ is not served tightly, causing at least $w_x w_y = (k+1)^2$ units of extra cost. Hence $\mathcal{S}$ cannot be a good schedule. □

Lemma 10. *Consider the gadget $\pi(v_i, v_j)$ in a good schedule $\mathcal{S}$. Let $L^\#$ denote the list state when the first request to d within the gadget is served. If $v_j \prec v_i$ holds in $L^\#$, then this incurs at least one extra unit of cost in this gadget.*

Proof. Using Lemma 4, it is not hard to see that in a tight gadget, the following relative orderings in $L^\#$ are needed: $v_i \prec c$, $c \prec d$, $d \prec v_j$. Indeed, if $c \prec v_i$ holds in $L^\#$, then by Lemma 4 and

Proposition 3 the projected cost of the pair $\{c, v_i\}$ is at least

$$\delta_{v_i c}(v_i^2 c v_i \underline{c} v_i c c v_i c c c v_i^3) = 5.$$

(The extra c induced by Lemma 4 is underlined.) Hence the projected cost is strictly larger than the lower bound from Lemma 8. The proof for the other pairs is similar:

$$\begin{aligned} d \prec c: &\qquad \delta_{cd}(cd\underline{d}ccdddccc) = 4 \\ v_j \prec d: &\qquad \delta_{v_j d}(v_j^p d \underline{v_j} d d d v_j^{8-p}) = 3 \qquad (p \in \{3,4\}) \end{aligned}$$

Hence, each pair incurs extra cost if it is not in the right order. But the three required orderings also imply $v_i \prec v_j$. Hence if $v_j \prec v_i$ holds, at least one of the three relative orders cannot hold, resulting in at least one extra cost unit. □

Lemma 11. *Let L^* be the list state of a good schedule $\mathcal{S}$ just after the last gadget of E^+ has been served and before the first gadget of E^- is served. If the gadget $\pi(v_i, v_j)$ is tight, then $v_i \prec v_j$ must hold in L^*.*

Proof. Let us first consider a gadget $\pi(v_i, v_j)$ belonging to E^-. Remember that in this case $i < j$ and $v_i \prec v_j$ in L_0 hold. The gadget $\pi(v_i, v_j)$ is served after reaching L^*. From Lemma 10 it follows that for $\pi(v_i, v_j)$ to be tight, $v_i \prec v_j$ has to hold when the first request to d takes place in $\pi(v_i, v_j)$. According to Lemma 9, this is possible in a good schedule only if $v_i \prec v_j$ did hold in all list states before. Therefore $v_i \prec v_j$ must hold in L^*.

In the second case, consider a gadget $\pi(v_i, v_j)$ belonging to E^+. Hence we have $i > j$ and $v_j \prec v_i$ in L_0. This time, $\pi(v_i, v_j)$ is served before reaching L^*. Again Lemma 10 implies that for $\pi(v_i, v_j)$ to be tight, $v_i \prec v_j$ has to hold when the first request to d takes place in $\pi(v_i, v_j)$. To achieve this, the pair has to change its relative order. According to Lemma 9, v_j cannot move ahead of v_i anymore after $\pi(v_i, v_j)$ has been served. Therefore $v_i \prec v_j$ must hold in L^*. □

With Lemma 11 available, proving Claim 7 is quite easy. The desired permutation P can be obtained from L^*, after removing the items c and d.

Lemma 11 proves that the gadgets which are tight in $\mathcal{S}$ do not belong to $\kappa(P)$. Since every non-tight gadget incurs at least one unit of extra cost, there can be at most k non-tight gadgets. Even if for all these gadgets $\pi(v_x, v_y)$ we have $v_y \prec v_x$ in P, it still holds $|\kappa(P)| \leq k$.

5 OLUP is NP-hard

We will now reduce WLUP to OLUP. The reduction is defined by a function g that converts a WLUP instance into an OLUP instance. Let the WLUP instance be (L_0, σ, W) with items x_i of weight w_i for $i = 1, \ldots, n$. The function g converts (L_0, σ, W) into an OLUP instance $(\hat{L}_0, \hat{\sigma})$ by textually replacing any occurrence of x_i in L_0 and σ by the block $\hat{x}_{i,1}\hat{x}_{i,2}\ldots\hat{x}_{i,w_i}$.

As a general comment on the notation in this section: all objects with a hat-accent, such as $\hat{L}_0$, $\hat{\sigma}$, and $\hat{W}$, will always refer to an instance involving the $\hat{x}_{i,j}$ items. Those without the hat-accent refer to instances involving the x_i items.

Most of the section will be concerned with proving the following lemma.

Lemma 12. *If the access cost defined in (3) is replaced by*

$$\sum_{k: x_k \prec x_i \text{ in } L} w_i w_k \Bigg) + \binom{w_i}{2}, \tag{7}$$

it holds that $WOPT(L_0, \sigma, W) = OPT(\hat{L}_0, \hat{\sigma})$.

The following example illustrates Lemma 12. Consider the WLUP instance $(L_0, \sigma, W) = ([x_1x_2x_3],\ x_3x_2x_2x_3, [1,2,2])$. If we apply the g function, we obtain the OLUP instance

$$([\hat{x}_{1,1}\hat{x}_{2,1}\hat{x}_{2,2}\hat{x}_{3,1}\hat{x}_{3,2}],\ \ \hat{x}_{3,1}\hat{x}_{3,2}\hat{x}_{2,1}\hat{x}_{2,2}\hat{x}_{2,1}\hat{x}_{2,2}\hat{x}_{3,1}\hat{x}_{3,2}). \tag{8}$$

In our example, the optimum schedule for the WLUP instance is to move to the list state $[x_2x_3x_1]$ before the first request and stay there. This schedule has update cost 4 and access cost $5+1+1+5=12$. The intuition about Lemma 12 is that one can obtain an optimal schedule for $(\hat{L}_0, \hat{\sigma})$ by applying the textual replacement rule from function g to the optimum schedule of (L_0, σ, W). In our example, this means that the optimal way to serve $(\hat{L}_0, \hat{\sigma})$ is to move to list state $[\hat{x}_{2,1}\hat{x}_{2,2}\hat{x}_{3,1}\hat{x}_{3,2}\hat{x}_{1,1}]$ and to stay there. It is easy to check that this schedule has also update cost 4 and access cost 12. This concludes the example.

Using Lemma 12, we are able to prove the main theorem of this paper stated in the introduction.

Proof of Theorem 1. It is not hard to see that with the original access cost from (3), the relationship between $WOPT$ and OPT described in Lemma 12 turns into

$$WOPT(L_0, \sigma, W) + \sum_{x_i \in L_0} \binom{w_i}{2} \cdot |\sigma_{x_i}| = OPT(\hat{L}_0, \hat{\sigma}). \tag{9}$$

The second term of (9) on the left hand side only depends on σ and W, but not on the optimal schedule. Hence the reduction also works with the original access cost.

For the reduction to be polynomial, we need the weights of the items x_i to be bounded by a polynomial in the number of items in L_0 and the length of the request sequence σ. But from the previous section we know that WLUP is NP-hard even if the weights of the items are bounded by $O(n^2)$ where n is the number of items. □

Lemma 12 will be proved by showing $WOPT(L_0, \sigma, W) \geq OPT(\hat{L}_0, \hat{\sigma})$ and $WOPT(L_0, \sigma, W) \leq OPT(\hat{L}_0, \hat{\sigma})$.

In order to see the first, we transform (as in the example above) an optimal schedule $\mathcal{S}$ for (L_0, σ, W) into a schedule $\hat{\mathcal{S}}$ for $(\hat{L}_0, \hat{\sigma})$ by using the same textual replacement rule as in g. That is, $\hat{\mathcal{S}}$ is obtained by applying the textual replacement rule to all list states of $\mathcal{S}$ (recall that a schedule is just a sequence of list states). The cost of $\hat{\mathcal{S}}$ is $WOPT(L_0, \sigma, W)$. This follows by comparing the access and update cost in both schedules: Accessing x_i in a list L of $\mathcal{S}$ translates to accessing all items $\hat{x}_{i,1} \dots \hat{x}_{i,w_i}$ in $\hat{L}$ of $\hat{\mathcal{S}}$ in turn. In order to access $\hat{x}_{i,j}$ in $\hat{L}$, one has to pass all $\hat{x}_{l,k}$ with $x_l \prec x_i$ in $\hat{L}$ plus all $\hat{x}_{i,k}$ with $k < j$. Summing up the cost for accessing all items $\hat{x}_{i,j}$, $j = 1, \dots, w_i$, we obtain (7). Concerning update costs, a paid exchange of two weighted items x_i and x_k translates to swapping the order of every pair of items $\{\hat{x}_{i,j}, \hat{x}_{k,l}\}$ in $\hat{\mathcal{S}}$. This costs $w_i \cdot w_k$ units, exactly as in (2). This completes the proof of the first inequality.

In order to prove $WOPT(L_0, \sigma, W) \leq OPT(\hat{L}_0, \hat{\sigma})$, we have to show that a schedule $\hat{\mathcal{S}}$ for $(\hat{L}_0, \hat{\sigma})$ can be turned into a schedule $\mathcal{S}$ for (L_0, σ, W) without increasing the cost. Remember that the OLUP instance $(\hat{L}_0, \hat{\sigma})$ is equivalent to the WLUP instance $(\hat{L}_0, \hat{\sigma}, \mathbf{1})$. That is, the WLUP instance where all items have weight one. Also, the schedule $\hat{\mathcal{S}}$ is a valid schedule for $(\hat{L}_0, \hat{\sigma}, \mathbf{1})$. The schedule $\hat{\mathcal{S}}$ remains a valid schedule for all instances $(\hat{L}_0, \hat{\sigma}, \hat{W})$. This holds even if some of the weights are zero. (By equations (7) and (2), items with weight zero incur neither access cost nor update cost.)

The existence of $\mathcal{S}$ is proved using the probabilistic method based on the following randomized construction: For each $1 \leq i \leq n$, choose uniformly at random one of the items $\hat{x}_{i,1}, \hat{x}_{i,2}, \dots, \hat{x}_{i,w_i}$ in block i. Let X be the set of chosen items.

Let $(\hat{L}_0, \hat{\sigma}, \hat{W})$ be the instance obtained by reweighing all the items such that $\hat{w}_{i,j} = w_i$ iff $\hat{x}_{i,j} \in X$ and $\hat{w}_{i,j} = 0$ otherwise. Without increasing the cost, one can obtain a schedule $\mathcal{S}$ for (L_0, σ, W) by replacing $\hat{x}_{i,j}$ by x_i if $\hat{x}_{i,j} \in X$ and removing $\hat{x}_{i,j} \notin X$ from the list states of $\hat{\mathcal{S}}$.

Notice that $\mathcal{S}$ does consist of $|\hat{\sigma}|$ list states. But only in the list states that correspond to a request to an item $\hat{x}_{i,j} \in X$ in the original schedule $\hat{\mathcal{S}}$, we will perform a request by accessing x_i. One could remove the list states without request from the schedule $\mathcal{S}$, without increasing the cost of the schedule, but the calculations are easier if we assume that the list passes through these states.

The projected cost $\gamma_{\hat{x}_{i,j}\hat{x}_{\ell,m}}(\hat{\sigma}, \hat{\mathcal{S}})$ will be abbreviated by $\gamma(\hat{x}_{i,j}, \hat{x}_{\ell,m})$ in this proof since in what follows $\hat{\mathcal{S}}$ and $\hat{\sigma}$ will be fixed. The cost of $\hat{\mathcal{S}}$ can be expressed as

$$\Gamma(\hat{\mathcal{S}}) = \sum_{\substack{\{\hat{x}_{i,j}, \hat{x}_{\ell,m}\} \\ i \neq \ell}} \gamma(\hat{x}_{i,j}, \hat{x}_{\ell,m}) + \sum_i \sum_{\substack{\{\hat{x}_{i,j}, \hat{x}_{i,m}\} \\ j \neq m}} \gamma(\hat{x}_{i,j}, \hat{x}_{i,m}).$$

The first sum represents the cost due to pairs $\{i, \ell\}$ $(i \neq \ell)$ and the second sum counts for each i the cost due to the pairs of items inside block i. (There are no visible weight factors since all weights are 1 in this instance.)

A property of the function g that will be useful in this proof is that for every block i,

$$\sum_{\substack{\{\hat{x}_{i,j}, \hat{x}_{i,m}\} \\ j \neq m}} \gamma(\hat{x}_{i,j}, \hat{x}_{i,m}) \geq \sum_{\substack{\{\hat{x}_{i,j}, \hat{x}_{i,m}\} \\ j \neq m}} \delta(\hat{x}_{i,j}, \hat{x}_{i,m}) = \binom{w_i}{2} \cdot |\sigma_{x_i}| \tag{10}$$

This follows from $\sigma_{\{\hat{x}_{i,j}\hat{x}_{i,m}\}} = (\hat{x}_{i,j}\hat{x}_{i,m})^{|\sigma_{x_i}|}$.

As we will show next, the expected cost of $\mathcal{S}$ is bounded by the cost of $\hat{\mathcal{S}}$. First, we have to introduce proper random variables. For each item $\hat{x}_{i,j}$, we introduce a random variable $z_{i,j}$. We set $z_{i,j} := 1$ if $x_{i,j} \in X$, and $z_{i,j} := 0$ otherwise.

Using the cost as defined in (7), the cost of the schedule $\mathcal{S}$ is then simply

$$\sum_{\substack{\{\hat{x}_{i,j}, \hat{x}_{\ell,m}\} \\ i \neq \ell}} \gamma(\hat{x}_{i,j}, \hat{x}_{\ell,m}) \cdot z_{i,j} z_{\ell,m} \cdot w_i w_m + \sum_{i,j} z_{i,j} \binom{w_i}{2} \cdot |\hat{\sigma}_{\hat{x}_{i,j}}|$$

This is a random variable, whose expectation is

$$\begin{aligned}
&E\left[\sum_{\substack{\{\hat{x}_{i,j}, \hat{x}_{\ell,m}\} \\ i \neq \ell}} \gamma(\hat{x}_{i,j}, \hat{x}_{\ell,m}) \cdot z_{i,j} z_{\ell,m} w_i w_m + \sum_{i,j} z_{i,j} \binom{w_i}{2} \cdot |\hat{\sigma}_{\hat{x}_{i,j}}|\right] \\
=&\sum_{\substack{\{\hat{x}_{i,j}, \hat{x}_{\ell,m}\} \\ i \neq \ell}} \gamma(\hat{x}_{i,j}, \hat{x}_{\ell,m}) \cdot E[z_{i,j} z_{\ell,m}] w_i w_m + \sum_{i,j} E[z_{i,j}] \binom{w_i}{2} \cdot |\sigma_{x_i}| \\
=&\sum_{\substack{\{\hat{x}_{i,j}, \hat{x}_{\ell,m}\} \\ i \neq \ell}} \gamma(\hat{x}_{i,j}, \hat{x}_{\ell,m}) \cdot E[z_{i,j}] E[z_{\ell,m}] w_i w_m + \sum_i \binom{w_i}{2} \cdot |\sigma_{x_i}| \sum_j E[z_{i,j}] \\
=&\sum_{\substack{\{\hat{x}_{i,j}, \hat{x}_{\ell,m}\} \\ i \neq \ell}} \gamma(\hat{x}_{i,j}, \hat{x}_{\ell,m}) + \sum_i \binom{w_i}{2} \cdot |\sigma_{x_i}| \\
\leq&\sum_{\substack{\{\hat{x}_{i,j}, \hat{x}_{\ell,m}\} \\ i \neq \ell}} \gamma(\hat{x}_{i,j}, \hat{x}_{\ell,m}) + \sum_i \sum_{\substack{\{\hat{x}_{i,j}, \hat{x}_{i,m}\} \\ j \neq m}} \gamma(\hat{x}_{i,j}, \hat{x}_{i,m}) \\
=&\Gamma(\hat{\mathcal{S}}).
\end{aligned}$$

Here we used linearity of expectation, $|\hat{\sigma}_{\hat{x}_{i,j}}| = |\sigma_{x_i}|$ by construction of g, the fact that $z_{i,j}$ and $z_{\ell,m}$ are independent for $i \neq \ell$, and $E[z_{i,j}] = 1/w_i$ and (10).

Since the expected cost of $\mathcal{S}$ is bounded by $\Gamma(\hat{\mathcal{S}})$, there must exist an assignment of the random variables $z_{i,j}$ such that the cost of the resulting schedule for (L_0, σ, W) is bounded by $\Gamma(\hat{\mathcal{S}})$.

6 Conclusions

We have shown that the Offline List Update Problem is *NP*-hard using a reduction from Minimum Feedback Arc Set. It seems to be quite difficult to use this technique to obtain an APX-hardness proof. For MINFAS instances with n vertices and m arcs, the optimal schedule for the WLUP instance (L_0, σ, W) created by the reduction has cost between $WOPT((L_0, \sigma, W))$ and $WOPT((L_0, \sigma, W)) + m$. Since $WOPT((L_0, \sigma, W)) = m\Theta(n^4)$ and $m = O(n^2)$, clearly this cannot lead to an APX-hardness proof. It would be very interesting to know whether other techniques allow an APX-hardness proof or whether a PTAS for OLUP exists.

Acknowledgments

The author likes to thank Bernd Gärtner, Bernhard von Stengel, Rolf Möhring and Rob van Stee for helpful discussions. The author is supported by Nuffield Foundation Grant NAL32608.

References

[1] S. Albers. Improved randomized on-line algorithms for the list update problem. *SIAM Journal of Computing*, 27(3):682–693, 1998.

[2] S. Albers, B. von Stengel, and R. Werchner. A combined bit and timestamp algorithm for the list update problem. *Information Processing Letters*, 56(3):135–139, 1995.

[3] S. Albers and J. Westbrook. Self-organizing data structures. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms*, volume 1442 of *Lecture Notes in Computer Science*, pages 13–51. Springer, 1998.

[4] C. Ambühl, B. Gärtner, and B. v. Stengel. Optimal lower bounds for projective list update algorithms. *ACM Trans. Algorithms*, 9(4):31:1–31:18, Oct. 2013.

[5] C. Ambühl, B. Gärtner, and B. von Stengel. A new lower bound for the list update problem in the partial cost model. *Theoretical Computer Science*, 268(1):3–16, 2001.

[6] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA, 1998.

[7] F. d'Amore, A. Marchetti-Spaccamela, and U. Nanni. The weighted list update problem and the lazy adversary. *Theoretical Computer Science*, 108(2):371–384, 1993.

[8] S. Divakaran. An optimal offline algorithm for list update. *CoRR*, abs/1404.7638, 2014.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[10] S. Irani. Two results on the list update problem. *Information Processing Letters*, 38(6):301–306, 1991. Corrected version appeared as Technical Report 96-53, ICS Department, U.C. Irvine, CA, USA 1996.

[11] N. Reingold and J. Westbrook. Off-line algorithms for the list update problem. *Information Processing Letters*, 60(2):75–80, 1996.

[12] N. Reingold, J. Westbrook, and D. D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11(1):15–32, 1994.

[13] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[14] B. Teia. A lower bound for randomized list update algorithms. *Information Processing Letters*, 47(1):5–9, 1993.