Offline List Update is NP-hard

Christoph Ambühl

Institute for Theoretical Computer Science, ETH Zürich, 8092 Zürich, Switzerland.
ambuehl@inf.ethz.ch

Abstract. In the offline list update problem, we maintain an unsorted linear list used as a dictionary. Accessing the item at position i in the list costs i units. In order to reduce access cost, we are allowed to update the list at any time by transposing consecutive items at a cost of one unit. Given a sequence σ of requests one has to serve in turn, we are interested in the minimal cost needed to serve all requests. Little is known about this problem. The best algorithm so far needs exponential time in the number of items in the list. We show that there is no polynomial algorithm unless $P = NP$.

Keywords. On-line algorithms, competitive analysis, list-update, NP.

1 Introduction

The *list update problem* is a classical online problem in the area of self-organizing data structures [16, 4, 8]. In this paper, we will be concerned with the *offline* version of that problem (*OLUP*). In both versions, requests to items in an unsorted linear list must be served by accessing the requested item. If the item is at position i in the list, the access incurs a cost of i units in the *full* cost model and $i - 1$ units in the *partial* cost model [10, 11]. The goal is to keep access cost small by rearranging the items in the list. We have so-called *free exchanges* (the requested item can be moved closer to the list at no charge just after the request) and *paid exchanges* (any two consecutive items can be transposed at cost one).

An *online* algorithm must serve the sequence σ of requests one item at a time, without seeing future requests. An *offline* algorithm knows the entire sequence σ in advance. A feasible (but not necessarily optimal) solution for an instance is called a *schedule*.

To measure the performance of an online algorithm A on some sequence σ, we compare its overall cost $A(\sigma)$ with the cost $OPT(\sigma)$ incurred by an optimal offline algorithm, and we call A c-competitive, if for a suitable b,

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

for all sequences σ. If A is randomized, $A(\sigma)$ denotes the expected cost incurred by serving σ.

It is known that the best randomized online algorithm satisfies $1.5 \leq c \leq 1.6$ in the full cost model [2, 17]. For the partial cost model, a lower bound of $c \geq 1.50084$ can be shown [5].

Because of (1), the analysis of an online algorithm requires some understanding of the optimal offline algorithm *OPT*. Most algorithms analyzed so far have a property called *projectivity* [7]. For a projective algorithm A, a proof that A is c-competitive on lists with only two items generalizes automatically to longer lists. Fortunately on two items, *OPT* is fully understood. Recently it was shown that the upper bound of 1.6 cannot be further improved by projective algorithms [6]. This means that understanding *OPT* has become an even more important issue.

A better understanding of *OPT* could be helpful in two respects for online list update. First and most obviously, it may lead to better lower bounds on $OPT(\sigma)$ to plug into (1); second, it might reveal structural properties that turn out to be useful in devising better online algorithms. Properties of *OPT* have already been studied in the past [14, 15, 2, 3].

The size of an *OLUP* instance on n items and m requests is $\Theta(\log(n) \cdot m)$. But we can assume $m \geq n$. Therefore, an algorithm is still polynomial if its runtime is polynomial in n.

The currently best algorithm for *OLUP* runs in $O(2^n n! m)$ [14]. It is based on a straightforward dynamic programming algorithm for metrical task systems [13] which works as follows.

Let $d(i, L)$ be the minimal cost needed to serve the first i requests of the request sequence σ and end up in the list state L. We define $d(0, L) = 0$ if L is the initial list state and $d(0, L) = \infty$ for all other L. We can compute $d(i, L)$ using

$$d(i, L) = \min_{L'} \big(d(i-1, L') + trans(L', L) + acc(i, L) \big) . \tag{2}$$

Here, $trans(L', L)$ denotes the minimal cost to move from state L' to L and $acc(i, L)$ denotes the cost for accessing σ_i in L.

Note that there is always an optimal algorithm which does no free exchanges, therefore $trans(L, L')$ is easy to compute. The time needed to compute all $d(i, L)$ is $O((n!)^2 m)$.

This runtime can be reduced to $O(2^n n! m)$ by using the fact that there is an optimal algorithm which uses only so-called *subset transfers* [14]. In a subset transfer, one moves a subset of the items preceding the requested item x just behind x without changing their relative orders. Only $O(2^n)$ among the $n!$ possible transformations are subset transfers.

In this paper, we show that it is *NP*-hard to decide whether a sequence σ can be served with cost less or equal k. Therefore, there cannot be a polynomial time algorithm in n and m unless *P* = *NP*. This solves a longstanding open question ([8], *page 22*). One interpretation of this result is that the structure of *OPT* is not easy to understand; consequently, it might be very difficult to develop (and analyze) non-projective list update algorithms that improve the current upper bound of 1.6.

As an important part of the proof, we introduce a generalization of *OLUP* called *weighted list update problem* (*WLUP*). Here the items have a weight that influences access and transposition cost. A version of *WLUP* was considered already in [9], but our definitions and applications are different.

Throughout this paper, we assume the partial cost model. This is simpler to analyze than the original full cost model. It is easy to obtain the value of $OPT(\sigma)$ in the full cost model by adding $|\sigma|$ to the optimal cost in the partial cost model. Therefore, the proof certainly holds for the full cost model as well.

2 The Weighted List Update Problem

In this section, we generalize offline list update to items with *weights*. These weights have to be positive integers. We denote weighted items by capital letters.

An instance (σ, L, W) of (*WLUP*) consists of a request sequence σ and an initial list L over a set of weighted items X_i with weights w_i. Let W be the vector whose ith entry w_i is the weight of item X_i.

The cost incurred by operating on weighted items are the following. In order to transpose two items X_i and X_j with weights w_i and w_j respectively, we pay $w_i \cdot w_j$ units. The access cost for an item X_i with weight w_i are the following. Let S be the set of items in front of X_i, then accessing X_i costs

$$\binom{w_i}{2} + \sum_{l:X_l \in S} w_i \cdot w_l \tag{3}$$

This setting is motivated by the following interpretation of *WLUP* in terms of *OLUP*. Think of a weighted item X_i as a vector of items $[x_{i,1} \dots x_{i,w_i}]$. Accessing X_i means to access every item in X_i's vector in turn. In order to access $x_{i,j}$, one has to pass all $x_{l,k}$ with $X_l \in S$ plus all $x_{i,k}$ with $k < j$. Summing up over all items in X_i, we get (3). If two weighted items X_i and X_j are transposed, every item in X_i's vector has to pass every item in X_j's vector. In total, we need $w_i \cdot w_j$ transpositions.

The next lemma shows that the optimal schedules for the *WLUP* instance and its interpretation as an *OLUP* instance have the same cost. The lemma also serves as a proof for the reduction

$$\textit{WLUP} \leq_p \textit{OLUP}. \tag{4}$$

Note that the reduction is polynomial only for instances where the weights of the items are polynomial in the number of items or requests.

Lemma 1. *Let (σ, L, W) be an instance of WLUP and let (σ', L') be the transformation of (σ, L, W) to OLUP by replacing the items of (σ, L) by vectors of items. Then the costs of their optimal schedules are the same.*

Proof. Let us assume that the items of (σ, L, W) are denoted by X_i, and let w_i be the weight of X_i. In order to obtain (σ', L'), we replace the X_i by $x_{i,1} \dots x_{i,w_i}$ in L and σ. As an example, $\sigma = X_2 X_2 X_1$, $L = [X_1 X_2 X_3]$, and $W = [3, 2, 2]$ transforms to

$$\sigma' = x_{2,1}x_{2,2}\, x_{2,1}x_{2,2}\, x_{1,1}x_{1,2}x_{1,3} \text{ and } L' = [x_{1,1}x_{1,2}x_{1,3}\, x_{2,1}x_{2,2}\, x_{3,1}x_{3,2}].$$

Clearly, the optimal cost of (σ', L') are not larger than those of (σ, L, W), because we can serve (σ', L') according to the interpretation given above. It remains to show the the optimal cost for (σ, L, W) are not larger than those of (σ', L').

From (σ', L'), we can produce an instance of *WLUP* called $(\sigma'', L'', \mathbf{1})$ which has the same optimal value as (σ', L'): In a first step, we replace the $x_{i,j}$ by weighted items $Y_{i,j}$ with weight $w_{i,j}$. Note that for the moment, we do not set values to the $w_{i,j}$. Let us refer to this instance by (σ'', L'', W''). In order to obtain $(\sigma'', L'', \mathbf{1})$, we further assign all weights to $w_{i,j} = 1$. An optimal schedule for (σ', L') is clearly also optimal for $(\sigma'', L'', \mathbf{1})$. (The cost for a pair of items with weight one in *WLUP* are the same as for a pair of items in *OLUP*.) Therefore, the cost of an optimal schedule is the same for both $(\sigma'', L'', \mathbf{1})$ and (σ', L').

We can apply the optimal schedule for $(\sigma'', L'', \mathbf{1})$ also to (σ'', L'', W'') with any setting of W''. Note that the schedule remains the same by changing the weights. Only the cost for serving the schedule change. The cost for serving this schedule can be expressed as a function in the $w_{i,j}$.

In order to do so, we define constants $f_{(i,j),(k,l)}$ for each pair $\{Y_{i,j}, Y_{k,l}\}$. In $f_{(i,j),(k,l)}$, we sum up the number of times $Y_{i,j}$ and $Y_{k,l}$ are transposed in the schedule plus the number of times $Y_{i,j}$ is in front of $Y_{k,l}$ when $Y_{k,l}$ is requested and vice versa. So these constants are independent of $w_{i,j}$ and $w_{k,l}$. The total cost of the optimal schedule for $(\sigma'', L'', \mathbf{1})$ applied to (σ'', L'', W'') then is

$$C = \sum_{i \leq k, j, l} f_{(i,j),(k,l)} \cdot w_{i,j} w_{k,l} + \sum_{i,j} \binom{w_{i,j}}{2} \cdot |\sigma''_{Y_{i,j}}|. \tag{5}$$

Here $|\sigma''_{Y_{i,j}}|$ denotes the number or requests to $Y_{i,j}$ in σ''. The second sum comes from the binomial coefficients in (3). The second term of (3) is part of the $f_{(i,j),(k,l)}$.

For any choice of the $w_{i,j}$, equation (5) upper bounds the cost of the optimal schedule for (σ'', L'', W''). This holds because (5) denotes the cost of a legal (but not necessary optimal) schedule for (σ'', L'', W'').

In order to prove $(\sigma'', L'', \mathbf{1}) \geq (\sigma, L, W)$, we proceed as follows. Starting with the initial values $w_{i,j} = 1$ for all i and j, we will change the values of the $w_{i,j}$ step by step in such a way that the value of (5) does not increase and we will end up with an instance which is equivalent to (σ, L, W).

This is how we change the weights. As long as there is an i with $w_{i,j} > 0$ and $w_{i,k} > 0$ for some $i \neq k$, we set one of them to the sum of the two. The other one is set to zero. Note that an item with weight zero incurs neither access nor transposition cost.

Let us write (5) such that we can more easily detect how its value changes if we change $w_{i,j}$ and $w_{i,k}$.

$$\begin{aligned} C = C_0 + C_1 \cdot w_{i,j} + C_2 \cdot w_{i,k} + f_{(i,j),(i,k)} \cdot w_{i,j} w_{i,k} \\ + \binom{w_{i,j}}{2} |\sigma''_{Y_{i,j}}| + \binom{w_{i,k}}{2} |\sigma''_{Y_{i,k}}| \end{aligned} \tag{6}$$

Here, C_0 denotes all costs independent of both $w_{i,j}$ and $w_{i,k}$. By $C_1 \cdot w_{i,j}$ and $C_2 \cdot w_{i,k}$, we denote cost depending linearly only on one of the two.

Assume w.l.o.g. $C_1 \leq C_2$. If we set the new value of $w_{i,j}$ to $w_{i,j} + w_{i,k}$ and set $w_{i,k}$ to zero, the value of (5) does not increase. This holds because C_0 does not change and

$$C_1(w_{i,j} + w_{i,k}) + C_2 \cdot 0 \leq C_1 \cdot w_{i,j} + C_2 \cdot w_{i,k}, \tag{7}$$

and furthermore

$$\begin{aligned} f_{(i,j),(i,k)} \cdot w_{i,j} w_{i,k} + \binom{w_{i,j}}{2} |\sigma''_{Y_{i,j}}| + \binom{w_{i,k}}{2} |\sigma''_{Y_{i,k}}| \geq \\ f_{(i,j),(i,k)} \cdot (w_{i,j} + w_{i,k}) \cdot 0 + \binom{w_{i,j} + w_{i,k}}{2} |\sigma''_{Y_{i,j}}| + \binom{0}{2} |\sigma''_{Y_{i,k}}|. \end{aligned} \tag{8}$$

Using $|\sigma''_{Y_{i,j}}| = |\sigma''_{Y_{i,k}}|$ and $f_{(i,j),(i,k)} \geq |\sigma''_{Y_{i,j}}|$, inequality (8) is straightforward.

This reweighting process must terminate because in each step, the number of $w_{i,j}$ set to zero increases by one. What we end up with is an instance where for each i we have exactly one $w_{i,j}$ with $w_{i,j} = w_i$, and all the other $w_{i,j}$ are zero. This instance is equivalent to (σ, L, W) we started with. Just rename the $Y_{i,j}$ with $w_{i,j} = w_i$ to X_i and forget about the $Y_{i,j}$ which have weight zero. Because we did not increase the value of (5) by changing the weights, we have a schedule for (σ, L, W) which has the same cost as the optimal schedule for (σ', L'). $\square$

3 A Lower Bound

A total ordering of n items in a list L is defined by the relative ordering of all $\binom{n}{2}$ pairs of items in L. For example, the list state $L' = [X_2 X_3 X_1]$ corresponds uniquely to the set

$$\{X_2 < X_1, X_3 < X_1, X_2 < X_3\}.$$

Instead of one list containing n items, we can represent the list state by two-item lists $L_{X_i X_j}$, for each pair of items in L. The state formed by these $\binom{n}{2}$ pair lists is legal if and only if we can combine the relative orders induced by the pair lists to a total ordering. Thus, the state

$$L_{X_1 X_2} = [X_1 X_2],\ L_{X_2 X_3} = [X_2 X_3],\ L_{X_1 X_3} = [X_3 X_1]$$

is not legal. But if we changed the ordering of the items in $L_{X_2 X_3}$, it would correspond to $[X_3 X_1 X_2]$.

We will serve our requests on the pair lists. In order to satisfy a request to an item X_i, we must access this item in all pair list it occurs. There are $n - 1$ of them. Hence, on a pair list $L_{X_i X_j}$, we serve only those requests in σ that go to X_i or X_j. We denote this subsequence by $\sigma_{X_i X_j}$ and call it the *projection* of σ to X_i and X_j. At any point in time, we are allowed to transpose the items in a pair list, paying $w_i \cdot w_j$ units. But we have to make sure that this leads to a legal state. If we access X_j in the list $[X_iX_j]$, we pay $w_i \cdot w_j$ units, but in $[X_jX_i]$ we pay noting. We call these access and transposition cost *pair costs*, because we can assign them uniquely to a pair if items, namely $\{X_i, X_j\}$. In addition to the access cost we already mentioned, we pay b units for each request, where $b = \binom{w}{2}$ if the requested item has weight w. We refer to these cost as *item cost*. Note that the item cost depend only on the request sequence, but not on the schedule. It can be easily seen from the definitions of the two models that they are equivalent.

From the pair list based model, we can derive a lower bound for *WOPT*, to which we will refer to as $\overline{WOPT}$ [2]. In order to do so, we forget about the condition that the state defined by the pair lists must correspond to a total ordering of all n items. So in $\overline{WOPT}$, any state of the pair lists is legal. This means that for each pair of items $\{X_i, X_j\}$, we can choose their relative order independently of the other pairs. In order to get the optimal schedule for the pair $\{X_i, X_j\}$, we just serve σ_{X_i,X_j} on L_{X_I,X_j} optimally. Thus, to derive $\overline{WOPT}$, we just have to solve $\binom{n}{2}$ *WLUP* instances on two items each. Fortunately, the algorithm described in the introduction runs in linear time if the number of items is fixed. Thus, $\overline{WOPT}$ is computable in polynomial time.

Note as well that the optimal schedule for an instance on two items does not depend on the weights, at least if both weights are strictly positive. This holds because of two reasons. First, the item cost are constant for every schedule of a given request sequence. Second, the pair cost are of the form $k \cdot (w_i \cdot w_j)$, where k does not depend on the weights.

Thus, the hardness of the problem comes from the total ordering that must hold at any time in the schedule of *WOPT*. Most of the time, the actual state $\overline{WOPT}$ is in will not be a legal one for *WOPT*. Therefore, some pairs will have to be served non-optimally in *WOPT*.

4 The Reduction

Lemma 1 allows us to show a reduction from an *NP*-complete problem to *WLUP* in order to prove *NP*-hardness of *OLUP*.

The *minimum feedback arc set problem* (*MINFAS*) [12] will serve well for that purpose. Its decision variant *MINFAS*(G, k) is defined as follows. Given a directed graph $G = (V, E)$ and $k \in \mathbf{N}$, we want to decide whether there is a subset $E' \subseteq E$ with $|E'| \leq k$ such that the Graph $G' = (V, E - E')$ is acyclic.

There is a second interpretation which is more natural for our purpose. We interpret an arc pointing from v_i to v_j as a constraint "v_i should be ahead of v_j". What we want to decide is whether there exists a total ordering of the vertices such that less than k constraints are unsatisfied.

We show a reduction from *MINFAS*(G, k) to the decision version of *WLUP*, denoted by *WLUP*(σ, L, W, k'). Here we want to decide whether there is a schedule which serves σ from the initial state L at maximal cost k'. More precisely, we give a function f that takes G and k as arguments and returns (σ, L, W) such that

$$\mathit{MINFAS}(G, k) \Leftrightarrow \mathit{WLUP}(\sigma, L, W, \overline{\mathit{WOPT}}(\sigma, L, W) + k). \tag{9}$$

From now on, we abbreviate the cost $\overline{WOPT}(\sigma, L, W)$ by $\overline{wopt}$. For the following section, it is important to understand how $\overline{WOPT}$ behaves in different situations. As $\overline{WOPT}$ treats all pairs of items independently, we have to investigate how sequences on two items are served optimally. Remember that in the two items case, the behavior does not depend on the weights, therefore it behaves like $\overline{OPT}$.

We consider a list containing only the items a and b. In order to describe $\overline{WOPT}$, we must find out in which cases one must, can or must not swap the two items. The following table gives the answer for a few cases, depending on how the remaining part of the request sequence looks like. We will encounter these cases later in this section and then refer to them by their number in $\langle \cdot \rangle$. The notation is analogous to the one for regular expressions. Thus, $(ba)^*$ denotes the empty sequence or any number of repetitions of ba. The sequence λ can be any sequence on a and b. If there is no λ at the end of the sequence, we assume that this is the end of the sequence.

must not swap:	$\langle 0 \rangle$	[ab]	$a\lambda$
	$\langle 1 \rangle$	[ab]	$baa\lambda$
	$\langle 2 \rangle$	[ab]	$(ba)^*$
can swap:	$\langle 3 \rangle$	[ab]	$babb\lambda$
	$\langle 4 \rangle$	[ab]	$(ba)^*b$
must swap:	$\langle 5 \rangle$	[ab]	$bbb\lambda$

We now describe the function f which transforms a *MINFAS* instance into a *WLUP* instance. For every vertex v_i of $G = (V, E)$, we have a weighted item V_i with weight $k + 1$. We call them *vertex items* and define $n := |V|$. Additionally, we have two items c and d both with weight one.

These are all the items we need. Let us check briefly that the weights are not too large in order to make (4) work. Clearly, the hard *MINFAS* instances obey $k < |E| \leq |V|^2$. Hence in those cases, the weights of the items are polynomial in the number of items. Thus, the reduction from *WLUP* to *OLUP* is polynomial.

We set $L = [V_1 V_2 V_3 \ldots V_n c\, d]$. The sequence σ is basically of the form

$$(V_1 V_2 V_3 \ldots V_n)^*,$$

with additional requests to c and d. It consists of two parts σ' and σ''. The first part is

$$\sigma' = V_1 V_2 V_3 \ldots V_n.$$

The second part consists of so called *arc gadgets*. An arc gadget γ for $(v_i, v_j) \in E$ looks as follows. Basically, it consists of 6 repetitions of σ', with additional requests to c and d. If $i < j$ the gadget looks as follows.

$$\gamma = V_1 ... V_{i-1} c\, V_i ... V_j d V_{j+1} ... V_i cc V_{i+1} ... V_{j-1} dddccc V_j ... V_n \sigma' \sigma' \sigma' \sigma' \tag{10}$$

For $j < i$, we have

$$\gamma = V_1 ... V_j ... V_{i-1} c\, V_i ... V_j d V_{j+1} ... V_i cc V_{i+1} ... V_{j-1} dddccc V_j ... V_i ... V_n \sigma' \sigma' \sigma' \tag{11}$$

The idea of γ is to make *WOPT* pay one unit extra if and only if V_j is behind V_i at the first request to d in the gadget.

Lets partition the set of arcs in G into two subsets. E^+ contains the arcs (v_i, v_j) with $i > j$, whereas E^- contains those with $i < j$. In σ'', we have one arc gadget for each arc in G, with the additional restriction that all the arc gadgets of the arcs in E^+ precede those in E^-.

In order to prove (9), we first look at some properties of this instance. We have seen in section 3 that the cost spent by $\overline{\mathit{WOPT}}$ on a pair of items lower bounds the cost *WOPT* needs for this pair. Therefore, in a schedule that costs no more than $\overline{\mathit{wopt}} + k$ units every pair of items involving a vertex item must be served according to $\overline{\mathit{WOPT}}$. This holds because the pair cost of such a pair are a multiple of $(k+1)$. Therefore, any non-optimal step involving a vertex item costs at least $k + 1$ additional units.

Let us first look at pairs consisting of two vertex items V_i and V_j, $i < j$. In the initial state, V_i is in front of V_j. Therefore, $\overline{\mathit{WOPT}}$ has to serve the following request sequence from initial state $[V_i V_j]$:

$$V_i V_j V_i V_j \dots V_i V_j \tag{12}$$

One way of optimally serving this instance is to do nothing at all. But there are even more optimal schedules for this sequence: In order to stay optimal, we are allowed to swap the two items exactly once (check $\langle 0 \rangle$, $\langle 2 \rangle$, and $\langle 4 \rangle$). Because one should never move V_j in front of V_i when the next request goes to V_i $\langle 0 \rangle$, this swap has to take place before a request to V_j.

It is easy to see that in an optimal schedule which pays less than $\overline{\mathit{wopt}} + k$ units, the list state before and after every gadget must have the sublist $[cd]$ at the end of the list state: Because of the three repetitions of σ' at the end of the edge gadgets and because of $\langle 5 \rangle$, the items c and d must surely be at the end of the list. If we now have a closer look at the requests to c and d only, we notice that a gadget ends up with three requests to c and starts with another one to c. Therefore, in order to be optimal, c has to be in front of d.

To see how *WOPT* serves the gadget for (v_i, v_j), we have to look at two cases. If V_i is in front of V_j, we can serve it with the same amount $\overline{\mathit{WOPT}}$ would pay. This is easy to check by showing that there is a schedule which serves each pair of items like $\overline{\mathit{WOPT}}$ would do: The crucial point is that in $\overline{\mathit{WOPT}}$, when the first request to d takes place, d must still be behind c $\langle 1 \rangle$, while c must be behind V_i $\langle 1 \rangle$ and d must be in front of V_j $\langle 5 \rangle$. Only if V_i is in front of V_j, we can fulfill these conditions. Note that c and d can pass V_k, $k \notin \{i, j\}$, but they do not have to $\langle 3 \rangle$. At the next request to c, we move c to the front of the list $\langle 3, 5 \rangle$. Later, at the first request of the request triple to d, we move d to the front as well $\langle 5 \rangle$, but c will have to pass d again later $\langle 5 \rangle$. Because of the additional ${\sigma'}^3$ finishing γ, both c and d must be moved behind all vertex items at the end without changing the relative order of c and d $\langle 5 \rangle$.

If V_i is behind V_j, not all the conditions mentioned in the previous paragraph can be fulfilled at the first request to d. The only way to fix this without paying more than k extra units is to move d in front of c at the first request to d and thus pay one unit more than $\overline{\mathit{WOPT}}$.

Now we are ready to prove (9). The easier part is the $\Rightarrow$ direction. If we can get an acyclic graph G' by removing only k arcs, we sort the vertices of G' topologically. The schedule which costs at most $\overline{wopt} + k$ looks as follows. We use the initial sequence σ' to rearrange the items V_i according to the topological order $\langle 4 \rangle$. For the rest of σ, we do not change the ordering of the vertex items anymore. Thus, we serve all vertex pairs optimally.

Concerning the arc gadgets, we can serve all those corresponding to the arcs in G' like $\overline{WOPT}$ would do it. For each arc we removed from G, we have to pay one unit extra. As there are at most k of them, we pay at most $\overline{wopt} + k$ units to serve σ.

It remains to prove the $\Leftarrow$ direction of (9). There are at most k gadgets whose cost where higher than those of $\overline{WOPT}$. We say a gadget is served *well* if we payed no more than $\overline{WOPT}$ to serve it. We will show that if we remove the arcs corresponding to those gadgets, the resulting graph will be acyclic.

Let the arcs of $C \subseteq E$ form a cycle in G. We have to prove that there is at least one arc gadget belonging to arcs in C which is not served well. For any arc $e = (v_i, v_j)$ and a list state L, we say e is *open* if we have V_i in front of V_j in L and *closed* otherwise. The arcs in $C \subseteq E^+$ are those which are closed in the initial list. In order to serve their gadget well, it has to be open when their gadget is served, but we cannot close them anymore $\langle 2 \rangle$. The arcs in $C \subseteq E^-$ are open in the initial list. If we want to serve them well, we can not close them before their gadget is served because we cannot reopen them $\langle 2 \rangle$.

Let us have a look at the list just after we served all arc gadgets for E^+ in σ. In order to serve all gadgets belonging to C well, all of them must be open at this time. This means for any arc $e = (v_i, v_j)$ in C that the item V_i must be in front of V_j in the current list. Because C forms a cycle, at least one of them must be closed and hence was not (if it belongs to E^+) or will not be (if it belongs to E^-) served well. This concludes the proof.

5 Conclusions

We have shown that the offline list update problem is *NP*-complete using a reduction from minimum feedback arc set to weighted list update. By changing the gadgets slightly, one can prove the same result also for the case where only free exchanges are allowed in order to update the list.

As an open question, it remains to show whether it can be decided in polynomial time whether

$$OPT(\sigma) = \overline{OPT}(\sigma).$$

Acknowledgements

The author likes to thank Bernd Gärtner, Bernhard von Stengel, and Rolf Möhring for helpful discussions.

References

1. S. Albers (1998), Improved randomized on-line algorithms for the list update problem. *SIAM J. Comput.* 27, no. 3, 682–693 (electronic). Preliminary version in *Proc. 6th Annual ACM-SIAM Symp. on Discrete Algorithms* (1995), 412–419.
2. S. Albers, B. von Stengel, and R. Werchner (1995), A combined BIT and TIMESTAMP algorithm for the list update problem. *Inform. Process. Lett.* 56, 135–139.
3. S. Albers (1998), A competitive analysis of the list update problem with lookahead. *Theoret. Comput. Sci.* 197, no. 1-2, 95–109.
4. S. Albers and J. Westbrook (1998), Self Organizing Data Structures. In A. Fiat, G. J. Woeginger, "Online Algorithms: The State of the Art", *Lecture Notes in Comput. Sci.*, 1442, Springer, Berlin, 13–51.
5. C. Ambühl, B. Gärtner, and B. von Stengel (2000), A new lower bound for the list update problem in the partial cost model. To appear in *Theoret. Comput. Sci.*
6. C. Ambühl, B. Gärtner, and B. von Stengel (2000), Optimal Projective Bounds for the List Update Problem. To appear in *Proc. 27th ICALP.*
7. J. L. Bentley, and C. C. McGeoch (1985), Amortized analyses of self-organizing sequential search heuristics. *Comm. ACM* 28, 404–411.
8. A. Borodin and R. El-Yaniv (1998), Online Computation and Competitive Analysis. Cambridge Univ. Press, Cambridge.
9. F. d'Amore, A. Marchetti-Spaccamela, and U. Nanni (1993), The weighted list update problem and the lazy adversary. *Theoret. Comput. Sci.* 108, no. 2, 371–384
10. S. Irani (1991), Two results on the list update problem. *Inform. Process. Lett.* 38, 301–306.
11. S. Irani (1996), Corrected version of the SPLIT
12. M. R. Garey, D. S. Johnson (1979), Computers and intractability. W. H. Freeman and Co, San Francisco.
13. M. Manasse, L. A. McGeoch, and D. Sleator (1988), Competitive algorithms for online problems. *Proc. 20nd STOC* (1988), 322-333.
14. N. Reingold, and J. Westbrook (1996), Off-line algorithms for the list update problem. *Inform. Process. Lett.* 60, no. 2, 75–80. Technical Report YALEU/DCS/TR-805, Yale University.
15. N. Reingold, J. Westbrook, and D. D. Sleator (1994), Randomized competitive algorithms for the list update problem. *Algorithmica* 11, 15–32.
16. D. D. Sleator, and R. E. Tarjan (1985), Amortized efficiency of list update and paging rules. *Comm. ACM* 28, 202–208.
17. B. Teia (1993), A lower bound for randomized list update algorithms, *Inform. Process. Lett.* 47, 5–9.