All Permutations Supersequence is coNP-complete

Przemysław Uznański

Helsinki Institute for Information Technology HIIT,
Department of Computer Science, Aalto University, Finland

July 10, 2015

Abstract

We prove that deciding whether a given input word contains as subsequence every possible permutation of integers $\{1, 2, \ldots, n\}$ is coNP-complete. The coNP-completeness holds even when given the guarantee that the input word contains as subsequences all of length $n-1$ sequences over the same set of integers. We also show NP-completeness of a related problem of *Partially Non-crossing Perfect Matching in Bipartite Graphs*, *i.e.* to find a perfect matching in an ordered bipartite graph where edges of the matching incident to selected vertices (even only from one side) are non-crossing.

1 Introduction and Preliminaries

The question of deciding for two words whether one is a *subsequence* of the other is one of the most basic problems in combinatorics. A folklore result states that a greedy solution is correct, and the problem is trivially in P. However, if we consider the related questions of finding the shortest common supersequence or the longest common subsequence (LCS), both have been shown to be NP-complete when allowed multiple input words, by Maier [12], improved to binary alphabets for LCS by Räihä and Ukkonen [16].

The question of constructing the shortest word containing as subsequences all permutations (so called *universal* words), was first posed by Knuth and attributed to Karp [6]. More precisely, writing $f(n)$ for the length of such a shortest word where n is the size of the alphabet, the question of determining the values of $f(n)$ was investigated (see [1] for known exactly values). Independently [2, 8, 11, 13, 14] provided an upper bound $f(n) \leq n^2 - 2n + 4$, while Newey [14] proved that this is tight for $n \leq 7$. A stronger upper bound $f(n) \leq \lceil n^2 - 7/3n + 19/3 \rceil$ has been recently provided by Radomirović [15]. Complementary, Kleitman and Kwiatkowski [9] have shown a lower bound of the form $f(n) \geq n^2 - C_\varepsilon n^{7/4+\varepsilon}$ for $\varepsilon > 0$.

In this paper, we investigate the problem of deciding whether a given sequence is universal. The question on the hardness of this problem was first, to our knowledge, posed by Amarilli [4]. We prove that this problem is coNP-complete, that is, a counterexample to the universality of any given word can be verified in polynomial time, but unless P= NP, no efficient algorithm exists to verify universality itself. Our result thus proves a separation between the problems of testing universality and testing whether an input word contains every word of given length (not only ones using distinct characters), with former being coNP-complete and the latter being in P by a simple greedy algorithm.*

In order to prove the coNP-hardness of *All Permutations Supersequence* (see Definition 1), we introduce the intermediate problem of *Locally Constrained Permutation* (see Definition 2), which captures the essential hardness of former, while itself being much easier to work with. This problem, where we ask to reconstruct a permutation given sets of available values for each given position, and a list of linear order constraints which every pair of consecutive positions has to satisfy, falls into a larger category of NP-complete problems

*The idea of the algorithm is as follows: iteratively take a letter of the alphabet for which the earliest occurrence after the current position is as far to the right as possible.

1	2	3	1	2	1	3
1	**2**	**3**	1	2	1	3
1	2	**3**	1	**2**	1	3
1	**2**	3	**1**	2	1	**3**
1	**2**	**3**	**1**	2	1	3
1	2	**3**	**1**	**2**	1	3
1	2	**3**	1	**2**	**1**	3

Figure 1: The sequence $1, 2, 3, 1, 2, 1, 3$ satisfies *All Permutations Supersequence* for the set $\{1, 2, 3\}$. The length of 7 is minimal.

involving permutation reconstruction. Similar problems were considered, *e.g.* permutation reconstruction from differences (De Biasi [5]) and recognizing sum of two permutations (Yu *et al.* [18]).

The reduction proving hardness of *Locally Constrained Permutation* can be shown to provide a very restricted instances. This fact, coupled with interpretation of permutations as perfect matchings in bipartite graphs, immediately provides us with a NP-hardness result for *Partially Non-crossing Perfect Matching in Bipartite Graphs* (see Definition 4). The problem of finding maximal non-crossing matching in bipartite graph, has been proposed and extended by Widmayer and Wong [17], and the problem itself reduces to longest increasing subsequence, which is solvable in polynomial time (Fredman [7]). Our result shows that lifting some non-crossing restrictions increases the computational complexity of the problem.

Notation. In this paper we will denote permutations using greek letter π. To ease the notation, we will use π both for the function $\pi\colon \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ and for the word $\pi = \pi_1\pi_2 \ldots \pi_n$. The set of all permutations of set $\{1, \ldots, n\}$ will be denoted as S_n. Given word w, we will write w^R to denote w reversed. For two words, we will write $w \sqsubseteq v$ meaning that w is a subsequence of v. Given a set S, a linear order on S is any binary relation $\prec$ such that for any two distinct $x, y \in S$ exactly one of $x \prec y$ or $y \prec x$ holds, and additionally $x \not\prec x$.

Now we are ready to formally define the problem that was already mentioned previously:

Definition 1. *All Permutations Supersequence:*

Input: *Integer $n > 0$, word T over alphabet $\{1, \ldots, n\}$.*

Question: *For every $\pi \in S_n$, does it hold that $\pi \sqsubseteq T$?*

An example of a shortest supersequence of all permutations of the set $\{1, 2, 3\}$ is provided in the Figure 1, together with placement of all permutations as subsequences.

2 Locally Constrained Permutation

First, we formally define the LCP problem.

Definition 2. *Locally Constrained Permutation:*

Input: *Integer $n > 0$, n sets $H_1, \ldots, H_n \subseteq \{1, \ldots, n\}$ and $n-1$ linear orders on $\{1, \ldots, n\}$: $\prec_1, \ldots, \prec_{n-1}$.*

Question: *Is there $\pi \in S_n$ such that for each $1 \le i \le n$: $\pi_i \in H_i$ and for each $1 \le i \le n-1$ we have $\pi_i \prec_i \pi_{i+1}$?*

To show the hardness of this problem, we will create an instance of LCP that encodes a given 3SAT instance. First, let us fix an instance of 3SAT, which consists of: m variables $v_1, \ldots, v_m$, and d clauses of form $\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$, where for each $1 \le i \le d, 1 \le j \le 3$: $\ell_{i,j} \in \{v_1, \ldots, v_m, \neg v_1, \ldots, \neg v_m\}$.

We observe that for any given i we don't need to fully specify the full order of $\prec_i$ on all of $\{1, \ldots, n\}$, as it is enough to specify it on $H_i \cup H_{i+1}$ only. We will also specify $\prec$ constraints not on every position, assuming it is possible for two consecutive positions to be under no constraint. Later we will show how to take care of this in LCP encoding.

The important property of LCP problem (and as well of any permutation reconstruction problem) is the fact that values can be used only once. Thus by assigning value to a position we are "blocking" this value from future use.

Literal encoding. We will encode each literal in a "memory cell" gadget. It consists of a separate position in the permutation with only two available choices, each of them corresponding to evaluating the underlying variable such that the literal evaluates to True or False, respectively. Thus, the information available can be "carried" over a long distance (to other occurrences of the same variable, or to positions evaluating truthfulness of the formula) by the fact that certain value is unblocked. However, this information is easily destroyed (one can think of it as read-once type of memory), thus we will need several working copies of the same memory cell.

Let $p \le 3d$ be the upper bound on the number of occurrences of a single variable in literals. Thus, for each literal $\ell_{i,j}$, there will be consecutive positions $\texttt{mem}(i,j), \texttt{mem}(i,j)+1, \ldots, \texttt{mem}(i,j)+p$, together with distinct values $\texttt{f}(i,j), \texttt{f}(i,j)+1, \ldots, \texttt{f}(i,j)+p$, $\texttt{t}(i,j), \texttt{t}(i,j)+1, \ldots, \texttt{t}(i,j)+p$, such that $H_{\texttt{mem}(i,j)} = \{\texttt{f}(i,j), \texttt{t}(i,j)\}, \ldots, H_{\texttt{mem}(i,j)+p} = \{\texttt{f}(i,j)+p, \texttt{t}(i,j)+p\}$. To enforce proper value copying, we set $\prec_{\texttt{mem}(i,j)}, \ldots, \prec_{\texttt{mem}(i,j)+p-1}$ such that:

$$\texttt{f}(i,j) \prec_{\texttt{mem}(i,j)} (\texttt{f}(i,j)+1) \prec_{\texttt{mem}(i,j)} \texttt{t}(i,j) \prec_{\texttt{mem}(i,j)} (\texttt{t}(i,j)+1),$$
$$(\texttt{f}(i,j)+1) \prec_{\texttt{mem}(i,j)+1} (\texttt{f}(i,j)+2) \prec_{\texttt{mem}(i,j)+1} (\texttt{t}(i,j)+1) \prec_{\texttt{mem}(i,j)+1} (\texttt{t}(i,j)+2),$$
$$\ldots$$
$$(\texttt{f}(i,j)+p-1) \prec_{\texttt{mem}(i,j)+p-1} (\texttt{f}(i,j)+p) \prec_{\texttt{mem}(i,j)+p-1} (\texttt{t}(i,j)+p-1) \prec_{\texttt{mem}(i,j)+p-1} (\texttt{t}(i,j)+p).$$

Observe, that there is a possibility for a one-sided error, that is assigning False to $\texttt{mem}(i,j)$ and True to $\texttt{mem}(i,j)+x$. However, those errors are not a problem for us, as they only occur when the literal is evaluated to False, thus the value of this literal is irrelevant to the evaluation of this clause in satisfying assignment.

Clause evaluation. For each clause we add a single position gadget verifying that the clause evaluates to True. Thus, for i-th clause, we have position $\texttt{clause}(i)$ such that $H_{\texttt{clause}(i)} = \{\texttt{f}(i,1), \texttt{f}(i,2), \texttt{f}(i,3)\}$. Thus, assigning value to position $\texttt{clause}(i)$ will be possible iff at least one of literals it contains is evaluated to True.

Variable values consistency. To make sure that different occurrences of the same variable are assigned the same value, we use a literal equality gadget. Furthermore, we will say that a literal is *positive* if it contains the simple variable, and is *negative* if it contains the negated variable.

We iterate over all pairs of literals using the same variable. Let $\ell_{i,j}$ and $\ell_{i',j'}$ be respectively the k-th and k'-th occurrences of this variable. We will be using the k'-th copy of "memory cell" gadget of $\ell_{i,j}$ and k-th copy of $\ell_{i',j'}$, thus making sure that each copy is used at most once for comparison.

Additionally, for any such pair of literals, there are two unique positions $\texttt{comp1}(i,j,i',j')$ and $\texttt{comp2}(i,j,i',j')$ such that:

- $H_{\texttt{comp1}(i,j,i',j')} = \{\texttt{t}(i,j)+k', \texttt{f}(i',j')+k\}$ and $H_{\texttt{comp2}(i,j,i',j')} = \{\texttt{f}(i,j)+k', \texttt{t}(i',j')+k\}$ if both literals are positive or both are negative;
- $H_{\texttt{comp1}(i,j,i',j')} = \{\texttt{t}(i,j)+k', \texttt{t}(i',j')+k\}$ and $H_{\texttt{comp2}(i,j,i',j')} = \{\texttt{f}(i,j)+k', \texttt{f}(i',j')+k\}$ otherwise.

The satisfying assignment to positions $\texttt{comp1}(i,j,i',j')$ and $\texttt{comp2}(i,j,i',j')$ is thus possible (in first case) iff both $\texttt{t}(i,j)+k'$ and $\texttt{t}(i',j')+k$ or both $\texttt{f}(i,j)+k'$ and $\texttt{f}(i',j')+k$ values were unblocked (and the second case works by analogy).

Balancing the number of positions and values. The above construction introduces n_1 possible values and n_2 positions for some $n_2 \leq n_1$. We introduce $n_2 - n_1$ new positions $\pi_{n_1+1}, \dots, \pi_{n_2}$ such that $H_{n_1+1} = \dots = H_{n_2} = \{1, \dots, n_2\}$, so that there exists a permutation $\pi \in S_{n_2}$ satisfying the larger instance iff there exists an injective function $\pi' \colon \{1, \dots, n_1\} \to \{1, \dots, n_2\}$ satisfying the original constraints.

Missing constraints. We also observe that we can choose not to impose any linear ordering restriction between any consecutive two positions i and $i+1$ by inserting a dummy position between them. Thus, all positions $i+1, \dots$ are moved one to the right, and a new dummy $i+1$ position is inserted. We also create a unique value c_i such that $H_{i+1} = \{c_i\}$ and $c_i \notin H_j$ for $j \neq i+1$, and and we construct the new linear orders $<_i$ and $<_{i+1}$ such that c_i is the largest value with respect to $\prec_i$ and the smallest value with respect to $\prec_{i+1}$.

Thus all missing linear order constraints can be taken care of iteratively, adding one extra position and value for each.

Lemma 2.1. *An instance of* 3SAT *is satisfiable iff a corresponding* LCP *instance is satisfiable.*

Proof. For the "only if" direction, it is clear by construction that a solution to the 3SAT instance can be used to construct a solution to the LCP instance.

To complete the proof, we need to show how one can reconstruct satisfying assignment to variables in 3SAT from a permutation π satisfying LCP instance.

Let us iterate over all the literals $\ell_{i,j}$. We will say that a literal is assigned TRUE (respectively FALSE) if for every $0 \leq k \leq p$ the corresponding position $\texttt{mem}(i,j)+k$ in π holds value $\texttt{t}(i,j)+k$ (respectively $\texttt{f}(i,j)+k$), and that its value is UNDECIDED otherwise.

Similarly, we will say that an occurrence of variable in literal is assigned value TRUE (respectively FALSE, UNDECIDED) if the literal is positive and is assigned the value of TRUE (respectively FALSE, UNDECIDED) or the literal is negative and is assigned the value of FALSE (respectively TRUE, UNDECIDED). We observe that for any two occurrences of the same variable, any configuration of values is allowed except one holding TRUE and another holding FALSE. However, as in the clause gadget we are using the first copy of any literal, any variable holding UNDECIDED is not helping to evaluate the clause to TRUE (the corresponding $\texttt{mem}(i,j)$ is assigned the value $\texttt{f}(i,j)$).

Consider assignment of values to variables of 3SAT as follow: if there exists occurrence holding TRUE or FALSE, we assign TRUE or FALSE, respectively, and otherwise we assign any value arbitrarily. By previous reasoning, this is a satisfying assignment to the given 3SAT instance.

□

Clearly, LCP is in NP and the reduction from 3SAT to LCP is constructed in polynomial time. Thus, by Lemma 2.1 we immediately get the following:

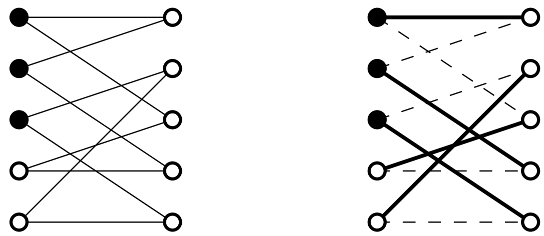

Figure 2: An example of an instance of *Partially Non-crossing Perfect Matching in Bipartite Graphs* (on the left). We require that matching edges incident to blacked nodes are non-crossing. A matching satisfying the constraints is on the right.

Proposition 2.2. LCP *is* NP-*complete.*

Observe that in the reduction from 3SAT to LCP we are explicitly using linear order constraints only for the memory cell gadget. Thus it is possible to define the values such that for any $1 \leq i \leq m, 1 \leq j \leq 3$:

$$\mathtt{f}(i,j) < (\mathtt{f}(i,j)+1) < \ldots < (\mathtt{f}(i,j)+p) < \mathtt{t}(i,j) < (\mathtt{t}(i,j)+1) < \ldots < (\mathtt{t}(i,j)+p)$$

and have every explicitly written $\prec_k$ be the ordering of integers. Additionally, we can make all the $\mathtt{mem}()$ positions smaller than any other ones, and possible values $\mathtt{f}(i,j)$ and $\mathtt{t}(i,j)$ are in the same order as for the corresponding positions, *i.e.* $\mathtt{mem}(i,j) < \mathtt{mem}(i',j')$ iff $\mathtt{t}(i,j) < \mathtt{f}(i',j)$. This way we make the reduction work so that the only linear order constraint used is monotonicity, it is used on some prefix of positions, and in this prefix even if there is no constraint, the possible values still satisfy monotonicity. Then, consider the following problem:

Definition 3. *Prefix Increasing Permutation:*

Input: *Integer $n > 0$, n sets $H_1, \ldots, H_n \subseteq \{1, \ldots, n\}$ and integer $0 \leq k \leq n$.*

Question: *Is there $\pi \in S_n$ such that for each $1 \leq i \leq n$: $\pi_i \in H_i$ and for each $1 \leq j \leq k$: $\pi_j < \pi_{j+1}$ are satisfied?*

It is worth noting, that both permutation reconstruction problems presented here have quite a natural interpretation in terms of matchings in bipartite graphs: each position i in permutation corresponds to some vertex a_i, and each value j corresponds to some vertex b_j, where we connect by edge (a_i, b_j) iff $j \in H_i$. Such a problem itself is naturally in P, however additional linear order constrains we impose transform it into NP-complete one.

For example, *Prefix Increasing Permutation* reduces to:

Definition 4. *Partially Non-crossing Perfect Matching in Bipartite Graphs:*

Input: *Ordered bipartite graph $G = (U, V, E)$ with orderings sets $U = (a_1, \ldots, a_n), V = (b_1, \ldots, b_n)$, and a subset $W \subseteq U$.*

Question: *Is there a perfect matching $M \subseteq E$, such that the restricted matching $M' = M \cap (W \times V)$ is non-crossing, i.e. if $(a_i, b_j), (a_k, b_l) \in M'$ then $i < j$ iff $k < l$?*

An example of an instance to this problem is presented on a Figure 2. We have the following immediate corollary of Proposition 2.2:

Corollary 2.3. Prefix Increasing Permutation *and* Partially Non-crossing Perfect Matching in Bipartite Graphs *are both* NP*-complete.*

3 All Permutations Supersequence

Now we are ready to show hardness of *All Permutations Supersequence* problem. We will do it by analyzing the complementary problem:

Definition 5. *Permutation Non-subsequence:*

Input: *Integer $n > 0$ and word T over alphabet $\{1, \ldots, n\}$.*

Question: *Is there $\pi \in S_n$ such that $\pi \not\sqsubseteq T$?*

Theorem 3.1. Permutation Non-subsequence *is* NP*-complete.*

Proof. Permutation Non-subsequence is clearly in NP. Thus, it is enough to construct a Preduction from LCP to *Permutation Non-subsequence*. Let us take an instance of LCP. Let us denote, given linear order $\prec$, by $\mathrm{ORD}(\prec) = i_1 i_2 \ldots i_n$ a word build from the permutation defining the order, that is $i_1 \prec i_2 \prec \ldots \prec i_n$. Similarly, given a set $H \subseteq \{1, \ldots, n\}$, let $\mathrm{ENC}(H) = i_1 \ldots i_{|H|}$ be an arbitrary word containing every element of H. We also denote by $\overline{H} = \{1, 2, \ldots, n\} \setminus H$.

Consider a word of the following form built from LCP instance:

$$W = \mathrm{ENC}(\overline{H_1})(\mathrm{ORD}(\prec_1)^R)\mathrm{ENC}(\overline{H_2})(\mathrm{ORD}(\prec_2)^R)\ldots(\mathrm{ORD}(\prec_{n-1})^R)\mathrm{ENC}(\overline{H_n}).$$

Clearly, W can be constructed in P.

We will show that for any given $\pi \in S_n$, π is a feasible solution to the LCP instance iff π is not a subword of W. Observe that each $\mathrm{ORD}(\prec_i)$ is a permutation of $\{1, \ldots, n\}$, so W clearly contains as a subword any word of length $n - 1$ (not necessarily a permutation).

To prove the *if* part, observe that if π is not a solution to the LCP instance, it must be for the following two reasons:

- For some i, $\pi_i \notin H_i$. We have $\pi \sqsubseteq W$ for the following reason: $\pi_j \in \mathrm{ORD}(\prec_j)^R$ for $1 \leq j < i$, $\pi_i \in \mathrm{ENC}(\overline{H_i})$ and $\pi_{j+1} \in \mathrm{ORD}(\prec_j)^R$ for $i \leq j < n$.
- For some i, $\pi_i \not\prec_i \pi_{i+1}$. But then π_i, π_{i+1} are exactly in this order in $\mathrm{ORD}(\prec_i)^R$, hence, matching π_j for $j < i$ and for $j > i$ as in the previous case.

To prove the *only if* part, let us take a $\pi \sqsubseteq W$. Let $i_1 < i_2 < \ldots < i_n$ be such that $\pi = W[i_1]W[i_2]\ldots W[i_n]$. At least one of the following conditions is fulfilled (as any subsequence contradicting both conditions at once can consist of positions from $\mathrm{ORD}(\prec_i)^R$, one position per value of i, thus has length $n - 1$ at most):

- There is j such that position i_j in W is part of $\mathrm{ENC}(\overline{H_j})$. But then $\pi_j \notin H_j$, meaning that π is not a solution to this LCP instance.
- There is j such that both i_j and i_{j+1} positions in W are part of $\mathrm{ORD}(\prec_j)^R$. But that implies $\pi_j \not\prec_j \pi_{j+1}$, thus π is not a solution to this LCP instance. ☐

As an immediate corollary of Theorem 3.1, we have:

Corollary 3.2. All Permutations Supersequence *is* coNP*-complete.*

4 Conclusion

We proved the hardness of a problem of deciding whether a sequence is universal to every permutation with respect to having as subsequence. Somehow related to testing universality of a sequence with respect to certain combinatorial structures are following open questions on hardness of testing whether a sequence is an *universal traversal sequence* (Aleliunas *et al.* [3]) and whether a sequence is an *universal exploration sequence* (Koucký [10]), that is whether a sequence defines a series of moves capable of exploring every (fixed size) connected graph.

Acknowledgements. We are grateful both to Antoine Amarilli and to Jukka Suomela for providing with valuable suggestions and for fruitful discussions.

References

[1] Online encyclopedia of integer sequences, sequence A062714. `http://oeis.org/A062714`. Accessed: 2015-06-08.

[2] L. Adleman. Short permutation strings. *Discrete Math.*, 10(2):197–200, Jan. 1974.

[3] R. Aleliunas, R. M. Karp, R. Lipton, L. Lovász, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *Foundations of Computer Science, 1979., 20th Annual Symposium on*, pages 218–223, Oct 1979.

[4] A. Amarilli. Recognizing sequences with all permutations of $\{1, \ldots, n\}$ as subsequences. `http://cstheory.stackexchange.com/questions/31559`. Accessed: 2015-06-08.

[5] M. D. Biasi. Permutation reconstruction from differences. *Electr. J. Comb.*, 21(4):P4.3, 2014.

[6] V. Chvátal, D. A. Klarner, and D. E. Knuth. Selected combinatorial research problems. Technical report, Stanford, CA, USA, 1972.

[7] M. L. Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11(1):29–35, 1975.

[8] G. Galbiati and F. P. Preparata. On permutation-embedding sequences. *SIAM Journal on Applied Mathematics*, 30(3):421–423, 1976.

[9] D. J. Kleitman and D. J. Kwiatkowski. A lower bound on the length of a sequence containing all permutations as subsequences. *J. Comb. Theory, Ser. A*, 21(2):129–136, 1976.

[10] M. Koucký. Universal traversal sequences with backtracking. *Journal of Computer and System Sciences*, 65(4):717–726, 2002. Special Issue on Complexity 2001.

[11] P. J. Koutas and T. C. Hu. Shortest string containing all permutations. *Discrete Math.*, 11(2):125–132, Jan. 1975.

[12] D. Maier. The complexity of some problems on subsequences and supersequences. *J. ACM*, 25(2):322–336, Apr. 1978.

[13] S. P. Mohanty. Shortest string containing all permutations. *Discrete Math.*, 31(1):91–95, Jan. 1980.

[14] M. Newey. Notes on a problem involving permutations as subsequences. Technical report, Stanford, CA, USA, 1973.

[15] S. Radomirović. A construction of short sequences containing all permutations of a set as subsequences. *Electr. J. Comb.*, 19(4):P31, 2012.

[16] K.-J. Räihä and E. Ukkonen. The shortest common supersequence problem over binary alphabet is NP-complete. *Theor. Comput. Sci.*, 16:187–198, 1981.

[17] P. Widmayer and C. K. Wong. An optimal algorithm for the maximum alignment of terminals. *Inf. Process. Lett.*, 20(2):75–82, 1985.

[18] W. Yu, H. Hoogeveen, and J. Lenstra. Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard. *Journal of Scheduling*, 7(5):333–348, 2004.