Concatenation versus Addition
in Knapsack Problems

Birgit Jenner

Report LSI-93-34-R
Concatenation versus Addition in Knapsack Problems

Birgit Jenner*
Fakultät für Informatik
Technische Universität München
80290 München, Germany
E-mail address: jenner@lsi.upc.es

October 22, 1993

Abstract

We consider the complexity of two knapsack problems that are defined with the operation "concatenation" instead of "addition". Whereas both addition problems are NP-complete, the complexity of the corresponding concatenation problems differ significantly. We show that one concatenation knapsack problem remains NP-complete while the other is NL-complete. We exhibit several related NP- and NL-complete problems. Furthermore, we investigate unary knapsack problems, presenting a unary knapsack that is contained in symmetric logspace and a close-to-unary knapsack that is NL-complete.

1 Introduction

One of the historical challenges of complexity theory is to compare operations, like addition, multiplication, concatenation, in terms of their complexity.

*On leave until March 1995, visiting Dept. Llenguatges i Sistemes Informàtics, Universitat Politècnica Catalunya, Pau Gargallo 5, E-08028 Barcelona, Spain, supported by a Habilitationstipendium of the Deutsche Forschungsgemeinschaft (DFG-Je 154 2/1).
"Is multiplication harder than addition?" is a famous question posed by Cobham [Co 65]. Recent approaches to this question show that addition is computable with constant-depth circuits (in AC⁰), while multiplication is not computable with such circuits [FSS 84]. Rather, multiplication is AC⁰-equivalent to majority and binary-count, and hence "complete" for the bigger class TC⁰ of constant-depth threshold circuits (see [CKV 84] [Bu 92]).

We are interested in the related question: "Is addition harder than concatenation?" Both operations are known to be in the complexity class AC⁰, i.e., are computable with constant-depth circuits. To answer our question one could intend to find a further classification of addition and concatenation within AC⁰, e.g., within the logtime hierarchy of Sipser [Si 83] that is a uniform version of AC⁰ [BIS 90]. We will take however a different approach. Instead of classifying the basic operations as such in terms of their complexity, we consider more involved addition problems and want to know whether their complexity decreases, when the problem is formulated with concatenation rather than addition, taking this as evidence that addition may be harder than concatenation.

This approach is partly motivated by the fact that an operation op, like concatenation, may be "characteristic" for a complexity class C in the sense that C has complete problems (or contains problems) that are essentially defined in terms of op. For example, consider the function class opt-L studied in [AJ 92] [AJ 93] that is a subclass of AC¹. As shown in [AJ 92], a complete problem for opt-L is iterated (MAX,·) matrix multiplication, where the operations maximum (MAX) and concatenation (·) replace the operations addition (+) and multiplication (*) of matrix multiplication. Iterated (MAX,+ matrix multiplication, in contrast, seems to be harder. It is known to lie in the class AC¹, but seems not to lie in the class opt-L nor in other subclasses of AC¹ (like e.g. SAC¹ [Ve 92]).

The addition problems that we consider in the following are two simple versions of the well-known knapsack problem:

<table>
<thead>
<tr>
<th>0-1 knapsack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given: a sequence w₁, w₂, ..., wₙ of (not necessarily different) positive integers, and a positive integer w,</td>
</tr>
<tr>
<td>Problem: is there a sequence of 0-1 valued variables x₁, x₂, ..., xₙ such that w = ∑ᵢ₌₁ⁿ xᵢ * wᵢ?</td>
</tr>
</tbody>
</table>
Knapsack with (nonnegative) repetition
as 0–1 knapsack, but \(x_1, x_2, \ldots, x_n\) may be arbitrary nonnegative integers.

It is well known that both of these problems are NP-complete (see [PS 82]
or [GJ 79], where 0–1 knapsack is called “subset sum”). Do these problems become easier when instead of addition, concatenation is considered? The resulting problems are:

\textbf{Concatenation knapsack (CK)}
Given: a string \(w\), and a sequence \(w_1, w_2, \ldots, w_n\) of (not necessarily different) strings (over alphabet \(\Sigma\)).
Problem: is there a sequence \(i_1, i_2, \ldots, i_k\) of pairwise distinct indices such that \(w = w_{i_1}w_{i_2}\ldots w_{i_k}\)?

\textbf{Concatenation knapsack with repetition (CKR)}
as concatenation knapsack, but the indices are not required to be distinct.

We show in Section 2 that CK is NP-complete. This is surprising, since the standard reductions from 3SAT to 0–1 knapsack via large “bit maps” make essentially use of the fact that addition can force the simultaneous presence of ones at different places in the map of the goal integer (see e.g. [Sch 92] or [PS 82], which uses \textsc{3-exact cover} instead of 3SAT, the satisfiability problem of Boolean formulas in conjunctive normal form with exactly three literals in each clause [GJ 79].) At first look, such a “communication” does not seem to be achievable via concatenation. Nevertheless, we present a reduction from 3SAT, where communication is achievable by letting the goal word force the joint choice of particular subsets of words from the given word sequence.

The concatenation knapsack problems CK and CKR are special cases of a coloured version of the well-known graph accessibility problem (GAP), that is NL complete [Jo 75]. We can reduce instances of the concatenation knapsack problem with repetition to GAP by constructing a graph whose vertex set is made up by the positions of the goal word \(w\), and whose edges indicate whether position \(i\) up to position \(j\) in \(w\) is covered by one of the words \(w_k\) of the given word sequence.

As a corollary of the NP-hardness of CK, we obtain the NP-completeness of the following problem:
**Rainbow GAP (RGAP)**

Given: a digraph $G = (V, E, C)$ with colours (from the set $C$) on the edges, $E \subseteq V \times C \times V$, and two designated vertices $s$ and $t$ in $V$.

Problem: is there a "rainbow path" from $s$ to $t$, i.e., a path that does not encounter any colour twice.

(Note that in such a graph there may be as many edges between two vertices $i$ and $j$ as there are colours in $C$.)

As another corollary of the NP-hardness of $CK$, also the following problem is shown to be NP-complete:

**Permutation Word Problem for Regular Expressions (PWREG)**

Given: a regular expression $R$ over alphabet $\Sigma$ and operators \{\cup, \cdot, \ast\} and strings $x_1, x_2, \ldots, x_n \in \Sigma^*$

Problem: is there a permutation $(i_1, i_2, \ldots, i_n)$ of the strings such that $x_{i_1}x_{i_2}\cdots x_{i_n} \in R$.

It turns out that $CK$ becomes considerably easier when repetitions are allowed. We show in Section 3 that concatenation knapsack with repetitions ($CKR$) is complete for nondeterministic logspace (NL-complete). As a corollary also the following restriction of PWREG is shown to be NL-complete:

**Ordered Sequence Word Problem for Regular Expressions (OSWREG)**

Given: a regular expression $R$ over alphabet $\Sigma$ and operators \{\cup, \cdot, \ast\} and strings $x_1, x_2, \ldots, x_n \in \Sigma^*$,

Problem: is there an ordered sequence $i_1 < i_2 < \ldots < i_k$ of indices such that $x_{i_1}x_{i_2}\cdots x_{i_k} \in R$?

With the NP-completeness on the one hand and the NL-completeness on the other, there is a significant difference in complexity of the two concatenation knapsack problems, while the analogous addition problems are both NP-complete. A similar decrease in complexity occurs for the unary versions of the addition knapsack problems, where the integers are given as a string of 1s. Both unary knapsack problems are solvable in NL. The standard explanation for this complexity jump focuses on the size and the consequently polynomial range of the given integers. Additionally, but less obviously, we are confronted here with the operation "concatenation": In the unary knapsack versions, addition is essentially concatenation. A still open question
is whether unary 0–1 knapsack is complete for NL [MS 80] [Co 85] [CH 88].
We show that when in the unary knapsack problem with repetitions the variables may also be negative integers, the problem is contained in symmetric logspace [LP 82], and hence unlikely to be complete for NL.

Finally in Section 4, we present a restriction of 0–1 knapsack that is NL-complete. Here all the given integers have the form \( w \cdot 2^p \) and are represented by the pair \((1^w, 1^p)\) of unary strings. We call this problem close-to-unary knapsack.

All the completeness results presented in the paper are via logspace-uniform NC\(^1\)-reductions, as can be easily verified. We do not further comment on the complexity of the reductions.

## 2 Concatenation Knapsack

We will show in this section that concatenation knapsack (\( CK \)) is NP-complete. By simple further reductions we also obtain the NP-completeness of several other problems defined in the introduction.

**Theorem 2.1** \( CK \) is NP-complete.

**Proof.** It is easy to see that \( CK \) is contained in NP, since it suffices to guess a sequence of indices \( i_1, i_2, \ldots, i_k, \ k \leq n, \) and to check whether \( w = w_{i_1}w_{i_2} \cdots w_{i_k}. \)

We show the NP-hardness of \( CK \) by an reduction from 3SAT, the satisfiability problem for boolean formulas in conjunctive normal form that have exactly 3 literals in each clause [GJ 79]. Let therefore \( X = \{x_1, x_2, \ldots, x_n\} \) be the set of variables and \( C = \{C_1, C_2, \ldots, C_m\} \) be the set of clauses of an arbitrary instance of 3SAT. We must construct a sequence \( W = w_1, w_2, \ldots, w_l \) of strings and a goal string \( w \) such that there is a set of indices \( i_1, i_2, \ldots, i_k, \) \( k \leq l, \) such that \( w = w_{i_1}w_{i_2} \cdots w_{i_k} \) if and only if \( C \) is satisfiable. The goal word \( w \) will consist of \( n + m \) substrings of which \( n \) are variable substrings, one for each variable \( x_i, \) and \( m \) are clause substrings, one for each clause \( C_j. \)

We denote these substrings by \( s_{x_i} \) and \( s_{C_j}, \) respectively. \( w \) is obtained by simply concatenating all variable and clause substrings:

\[
w := s_{x_1}s_{x_2} \cdots s_{x_n}s_{C_1}s_{C_2} \cdots s_{C_m}.
\]
We furthermore associate with each variable $x_i$ and each clause $C_j$ a set of strings, $W_{x_i}$ and $W_{C_j}$, respectively. All these strings together make up the string sequence:

$$W := (W_{x_i}, W_{C_j})_{1 \leq i \leq n, 1 \leq j \leq m}.$$ 

Let $\text{pair}$ be a simple pairing function for strings $i, j \in \{0, 1\}^*$ (computable in logspace-uniform NC$^1$). Then any of the strings in $W$ will be composed of the separator symbol $\#$ and substrings in $(0, 1)(0, 1)^*$ defined using $\text{pair}$ as follows:

For all $1 \leq i \leq n, 1 \leq j \leq m$, if literal $x_i$ appears in clause $C_j$, let

$$[ij] := \text{pair}(i, j)\#,$$

and if literal $\overline{x_i}$ appears in clause $C_j$, let

$$[ij] := 0\text{pair}(i, j)\#.$$

For a variable $x_i$, let $1 \leq j_1 < j_2 < \ldots < j_r$ be all clauses in which the literal $x_i$ appears, and $1 \leq \overline{j}_1 < \overline{j}_2 < \ldots < \overline{j}_r$ be all clauses in which $\overline{x_i}$ appears. The variable substring $s_{x_i}$ associated with $x_i$ then is:

$$s_{x_i} := \#[i_{j_1}][i_{j_2}]\cdots[i_{j_r}]\#[i_{\overline{j}_1}][i_{\overline{j}_2}]\cdots[i_{\overline{j}_r}]\#;$$

and the sequence of strings $W_{x_i}$ associated with the variable $x_i$ is:

$$W_{x_i} := ([i_{j_1}], [i_{j_2}], \ldots, [i_{j_r}], [i_{\overline{j}_1}], [i_{\overline{j}_2}], \ldots, [i_{\overline{j}_r}], \#, \#[i_{j_1}][i_{j_2}]\cdots[i_{j_r}]\#), \#[i_{\overline{j}_1}][i_{\overline{j}_2}]\cdots[i_{\overline{j}_r}]\#).$$

For a clause $C_j$ with literals $l_{1j}, l_{2j}, l_{3j}$, the clause substring $s_{C_j}$ is defined as follows:

$$s_{C_j} := \#[l_{1j}]\#[l_{2j}]\#[l_{3j}]\#,$$

where $[l_{pj}] = \begin{cases} [ij], & \text{if } x_i \text{ is the } p\text{th literal in } C_j; \\ [i\overline{j}], & \text{if } \overline{x_i} \text{ is the } p\text{th literal in } C_j. \end{cases}$

The word sequence for the clause $C_j$ is:

$$W_{C_j} := (\#, \#[l_{1j}]\#, \#[l_{3j}]\#, \#[l_{1j}]\#[l_{2j}]\#, \#[l_{2j}]\#[l_{3j}]\#).$$

The construction is such that any string $[ij]$ (respectively, $[i\overline{j}]$) occurs exactly twice in $w$, once in the variable substring $s_{x_i}$ and once in the clause
substring \( s_{C_j} \). Furthermore, to cover any variable substring \( s_x \), there are exactly two possibilities, using the strings

(v1) \([i_1j_1], [i_2j_2], \ldots, [i_nj_n]\) and the two strings \#,[i_1j_1][i_2j_2] \ldots [i_nj_n]\#, or

(v2) \([i_1j_1], [i_2j_2], \ldots, [i_nj_n]\) and the two strings \#,#[i_1j_1][i_2j_2] \ldots [i_nj_n]\#.

To cover any clause substring \( s_{C_j} \), there are exactly three possibilities, one for each literal \( l_{1j}, l_{2j}, l_{3j} \) of \( C_j \):

(c1) \([l_{1j}], \#, [l_{2j}]\#,[l_{3j}]\#\), or

(c2) \([l_{2j}], \#, [l_{1j}]\#,[l_{3j}]\#\), or

(c3) \([l_{3j}], \#, [l_{1j}]\#,[l_{2j}]\#\). #.

Note that the strings in \( W_{C_j} \) (except \#) do not fit at other places in \( w \).

We claim that \( w \) can be covered by strings of \( W \) if and only if \( C \) is satisfiable. First, assume that \( w \) can be covered by a sequence of strings from \( W \). Let \( v \) be the assignment for \( X \) that sets for all \( 1 \leq i \leq n \) \( v(x_i) = \text{TRUE} \), if the strings (v1) are used to cover \( s_{x_i} \), and \( v(x_i) = \text{FALSE} \), in the other case (v2). Because of (c1) up to (c3), for each clause \( C_j \) there is a substring \([l_{pj}]\) that must have been used to cover \( s_{C_j} \). By construction of \([l_{pj}]\), if \([l_{pj}] = [i_j]\), then \( x_i \) is the \( p \)th literal in \( C_j \) and \( v(x_i) = \text{TRUE} \), because otherwise the string \([i_j]\) would have been used to cover \( s_{x_i} \). Analogously, if \([l_{pj}] = [i_j]\), then \( \overline{x_i} \) is the \( p \)th literal in \( C_j \) and \( v(x_i) = \text{FALSE} \), because otherwise the string \([i_j]\) would have been used to cover \( s_{x_i} \). Hence the literal \( l_{pj} \) is set true by \( v \) for any clause \( C_j \), i.e., \( v \) satisfies \( C \).

Conversely, suppose \( C \) is satisfiable and let \( v : X \longrightarrow \{\text{TRUE, FALSE}\} \)
be any satisfying assignment for \( C \). Then we obtain a cover of any variable substring \( s_x \) of \( w \) by choosing the strings (v1), if the variable \( x_i \) is set \( \text{TRUE} \) by \( v \), and the strings (v2), otherwise. Note that any string \([i_j]\) (respectively, \([i_j]\)) not chosen so far corresponds to a literal \( x_i \) (respectively, \( \overline{x_i} \)) set true by \( v \). To cover each clause substring \( s_{C_j} \), we choose a string \([l_{pj}]\) that corresponds to a literal \( p \) of clause \( C_j \), that is set true by \( v \) (such strings are left untouched after our first choices), and depending on whether \( p \in \{1, 2, 3\} \) the two further strings of c1, c2, or c3. Hence, we have obtained a cover for the complete string \( w \).

By reducing concatenation knapsack to rainbow \( GAP (RGAP) \), we obtain \( NP \)-completeness for a simple variant of the \( NL \)-complete graph accessibility problem.
Corollary 2.2 \textit{RGAP} is NP-complete.

Proof. It is easy to see that \textit{RGAP} is contained in NP, since it suffices to guess a path \( p \) from \( s \) to \( t \) and to check whether on \( p \) no colour appears twice. Furthermore, as mentioned in the introduction, \textit{CK} can be reduced to \textit{RGAP} as follows. Let \( I = w, w_1, w_2, \ldots, w_n \) be an instance of the concatenation knapsack problem, where \( m \) is the length of the string \( w \). Then we express the property “string \( w_k \) is the subword of \( w \) formed by the bits of position \( i + 1 \) up to position \( j \)” as an edge labelled \( k \) between the edges \( i \) and \( j \) of a digraph \( G \) with vertices for the positions 0 up to \( m \). Clearly, \( I \in \textit{CK} \), i.e., the string \( w \) can be decomposed as \( w_{i_1} w_{i_2} \ldots w_{i_k}, 1 \leq k \leq n \), for pairwise distinct indices, if and only if there is a rainbow path with colours \( i_1, i_2, \ldots, i_k \) from vertex 0 to vertex \( m \) visiting vertices \( 0, l_1, l_2, \ldots, l_k = m \) in \( G \), where \( l_j - l_{j-1} \) denotes the length of \( w_{i_j} \) for \( 1 \leq j \leq k \). \hfill \Box

Using similar constructions as in the preceding proof, other variations on \textit{NL-complete} problems result in \textit{NP-complete} problems. For example, instead of a graph we may construct a rightlinear (or leftlinear) grammar. Then the \textit{NL-complete} word and emptiness problem for such grammars become \textit{NP-complete}, when only the application of rules with different terminal symbols is allowed.

\begin{center}
<table>
<thead>
<tr>
<th>Nonrepetitive derivation in linear grammars (\textit{NDLIN})</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Given:</strong> a rightlinear (or leftlinear) grammar ( G = (N, T, R, S) ) with rules ( A \rightarrow xB, x \in \Sigma^<em>, A \in N, B \in N \cup {\lambda} ), and a string ( w \in \Sigma^</em> ),</td>
</tr>
<tr>
<td><strong>Problem:</strong> can ( w ) be generated by ( G ) applying only rules that do not contain the same terminal word?</td>
</tr>
</tbody>
</table>
\end{center}

\begin{center}
<table>
<thead>
<tr>
<th>Non-emptiness for nonrepetitive derivation in linear grammars (\textit{\neg \emptyset NDLIN})</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Given:</strong> a rightlinear (or leftlinear) grammar ( G = (N, T, R, S) ) with rules ( A \rightarrow xB, x \in \Sigma^*, A \in N, B \in N \cup {\lambda} ),</td>
</tr>
<tr>
<td><strong>Problem:</strong> is there a string that is generated by ( G ) applying only rules that do not contain the same terminal word?</td>
</tr>
</tbody>
</table>
\end{center}

Corollary 2.3 \textit{NDLIN} and \( \neg \emptyset \textit{NDLIN} \) are NP-complete. \hfill \Box

As mentioned in the introduction, the permutation word problem for regular expression (\textit{PWREG}) is another \textit{NP-complete} problem closely related to concatenation knapsack.
Corollary 2.4 \textit{PWREG} is \textit{NP}-complete.

\textbf{Proof.} It is not hard to see that \textit{PWREG} is contained in \textit{NP}, since we can guess a permutation \( p \) of the given strings and then simulate an algorithm for the problem \( "p \in R" \). This problem is solvable in \textit{NL}, and hence in \textit{P}.

For \textit{NP}-hardness, we reduce from \textit{CK}. For strings \( w, w_1, \ldots, w_n \) over the alphabet \( \Sigma \), we set \( R := w \# \Sigma^* \), where \( \# \) is a symbol not contained in \( \Sigma \). Clearly, then we have \( w, w_1, \ldots, w_n \in \textit{CK} \) if and only if \( w_1, \ldots, w_n, \#, R \in \textit{PWREG} \), since all the strings not appearing in a cover of \( w \) are mapped to the \( \Sigma^* \) part of \( R \). \( \square \)

The problem remains \textit{NP}-complete when \( R \) is a context-free grammar, or when instead of a permutation a sequence \( i_1, i_2, \ldots, i_k \) of up to \( n = k \) pairwise distinct indices is required.

3 Knapsack Problems with Repetitions

In this section we will see that the complexity of concatenation knapsacks with repetitions may be significantly lower than the complexity of the corresponding problem without repetitions. First, we show that (general) concatenation knapsack is \textit{NL}-complete by reducing a topological sorted graph accessibility problem to it.

\textbf{Theorem 3.1} \textit{CKR} is \textit{NL} complete.

\textbf{Proof.} For the containment in \textit{NL}, let \( w, w_1, \ldots, w_n \) (strings over \( \Sigma \)) be an instance of \textit{CKR} such that \( w = b_1b_2 \cdots b_m \), \( b_i \in \Sigma \). Construct a nondeterministic machine \( M \) that uses a pointer \( p \) operating on \( w \), and a counter with maximal value \( m \). Both the pointer and the counter are initially set to 0. \( M \) repeatedly guesses indices \( i \) for \( 1 \leq i \leq n \). After each guess \( i \), \( M \) increases the counter and compares the string \( w_i \) with the substring of \( w \) with length \( |w_i| \) following the pointer, i.e., the string \( b_{p+1}b_{p+2} \cdots b_{p+|w_i|} \). If the two strings are not the same, \( M \) rejects, else \( M \) guesses the next index. \( M \) accepts when the pointer \( p \) reaches position \( m \), or, rejects when the counter has reached its maximal value \( m \) and still \( p < m \). Clearly, \( M \) accepts \textit{CKR}. Furthermore, \( M \) uses only logarithmic space (in the input length) for pointer, counter and bit-by-bit comparison. Hence, \textit{CKR} can be solved in \textit{NL}. 
For the completeness, we reduce the topological sorted version of the graph accessibility problem (TOPGAP) that is known to be NL-complete to CKR. TOPGAP is the problem: given a digraph $G = (V, E)$ with $V = \{0, 1, 2, \ldots, n\}$ and edge set $E$ such that if $(i, j) \in E$ then $i < j$, determine whether there is a path from 0 to $n$ in $G$. Given an instance $G$ of TOPGAP, define the corresponding knapsack instance with $w := 1$s$2$s$\ldots$n$s$, and for $e_k = (i, j) \in E$, let $w_k := i + 1$s$\ldots$j$s$. Note that, since $i < j$, $w_k$ will always contain the substring $j$s as suffix. Then, if there is a path $(0, i_1), (i_1, i_2), \ldots, (i_t, n)$ from 0 to $n$ in $G$, the concatenation of the strings that correspond to the edge sequence, i.e., $1$s$\ldots$i_1$s$, $i_1 + 1$s$\ldots$i_2$s$, $\ldots$, $i_{t-1}s$\ldots$n$s$, yield exactly $w$. Conversely, if $w$ can be produced by a concatenation of such subwords, then there is a path from 0 to $n$ in $G$ that passes (in that order) through the vertices that occur in each string just before the last $s$ symbol.

Using the proof of Theorem 3.1, it is not hard to show that concatenation knapsack (without repetition) is NL-complete when the sequence of indices underlying the wordcover is ordered. (The instance of CKR constructed in the hardness proof does not have a solution when words are repeated, and we only have to make sure that the strings $w_k$ are produced in prefix order by the reduction.) More precisely, the following variant of CK is NL-complete:

<table>
<thead>
<tr>
<th>Ordered concatenation knapsack (OCK)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given: a sequence $w_1, w_2, \ldots, w_n$ of strings (over alphabet $\Sigma$) and a string $w \in \Sigma^*$,</td>
</tr>
<tr>
<td>Problem: is there an ordered sequence of indices $i_1 &lt; i_2 &lt; \ldots &lt; i_k$ such that $w = w_{i_1}w_{i_2}\ldots w_{i_k}$?</td>
</tr>
</tbody>
</table>

**Corollary 3.2** OCK is NL-complete. □

As a second corollary, we obtain an NL-complete variant of the (general) word problem for regular expressions WREG: given a string $w$ and a regular expression $R$, decide whether $w \in R$. WREG is complete for NC$^1$ [Ba 89]. The variant is the problem OSWREG (see the Introduction), obtained by requiring in the NP-complete problem PWREG (Theorem 2.4) instead of a permutation the existence of an ordered sequence of indices.
Corollary 3.3 \( \text{OSWREG} \) is NL-complete.

**Proof.** It is not hard to see that \( \text{OSWREG} \in \text{NL} \). Hardness follows with Corollary 3.2, since \( \text{OCK} \) is the restriction of \( \text{OSWREG} \) with \( R = w \). \( \square \)

Note that we may substitute in \( \text{OSWREG} \) for "ordered sequence of indices" just "sequence of indices" (with \( k \) not necessarily bounded by \( n \)) and allow repetitions, and the problem remains NL-complete.

Unary 0–1 knapsack is the restriction of 0–1 knapsack (or, equivalently, concatenation knapsack with repetition) to a one-letter alphabet. Unary 0–1 knapsack is known to be solvable with nondeterministic logspace, but remains still an open problem whether it is NL-complete [MS 80] [Co 85] [CH 88].

We will show in the following that although most knapsack variants remain NP-complete, the corresponding unary versions may differ in complexity. Consider the following variant of 0–1 knapsack with repetition, where the variables may be arbitrary integers instead of nonnegative integers:

<table>
<thead>
<tr>
<th>Knapsack with repetition (KR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given: a sequence ( w_1, w_2, \ldots, w_n ) of positive integers, and a positive integer ( w ),</td>
</tr>
<tr>
<td>Problem: is there a sequence of integers ( x_1, \ldots, x_n ) such that ( w = \sum_{j=1}^{n} x_j \cdot w_j )?</td>
</tr>
</tbody>
</table>

Like the other variants of 0–1 knapsack, \( KR \) is NP-complete (see [PS 82]). The following theorem shows that the unary version is contained in symmetric logspace (SL) [LP 82], and hence, not likely to be complete for NL.

**Theorem 3.4** Unary \( KR \) is contained in SL.

**Proof.** We will reduce unary \( KR \) to \( \text{UGAP} \), the graph accessibility problem for undirected graphs that is SL-complete [LP 82]. First note that \( CKR \) is a special case of integer linear programming (ILP). In [PS 82] (Theorem 13.4) it is shown that if \( CKR \) has a feasible solution, then it has a feasible solution \( x_1, \ldots, x_n \) such that for all \( 1 \leq i \leq n \), \( x_i \) is bounded by the value \( b = n \cdot (\max w)^\delta (1 + w) \), where \( \max w \) is the maximum of the \( w_i \), \( 1 \leq i \leq n \).

Hence, given an instance \( 1^{w_1}, 1^{w_2}, \ldots, 1^{w_n} \) of the unary knapsack, \( b \) is polynomially bounded in its length, and we can construct an undirected graph \( G = (V, E) \) with vertices \( V = \{0, 1, 2, \ldots, B^3\} \) and edges \( E = \{(i, j) \mid \exists w_k, 1 \leq k \leq n : |i - j| = |w_k| \} \). It is not hard to see that there is a path
from vertex 0 to vertex \( w \) in \( G \) if and only if there is a solution of the unary knapsack. \( \square \)

It would be interesting to see more results concerning the complexity of other unary knapsack variants like, e.g., unary knapsack with nonnegative repetition (unary \( KNR \)). Having in mind the jump from \( \text{NP} \) to \( \text{NL} \) for concatenation knapsack to its unary version, one would expect unary \( KNR \) to be significantly easier than its corresponding nonunary version, shown to be \( \text{NL-complete} \) in Theorem 3.1.

4 A Knapsack for NL

If we restrict the \( \text{NP-complete} \) 0–1 knapsack to superincreasing sequences of integers we obtain a \( \text{P-complete} \) problem, as is shown in [KR 89]. (A sequence \( w_1, \ldots, w_n \) of positive integers is superincreasing, if each \( w_i \) is larger than the sum of the previous integers in the sequence, i.e., for each \( w_i, 1 \leq i \leq n, \) it holds \( w_i > \sum_{j=1}^{i-1} w_j. \)) On the other hand, unary codification of the integers yields a problem, namely unary 0–1 knapsack, that seems to lack the structure of \( \text{NL-complete} \) problems (see the discussion in the previous section). What restrictions must be posed on the knapsack values to obtain an \( \text{NL-complete} \) problem?

In this section, we answer this question. We present a restriction of 0–1 knapsack that is almost unary and \( \text{NL-complete} \). It presupposes that the positive integers have the form \( w \ast 2^p \) and are represented by the pair \( (1^w, 1^p) \) of unary strings. (Hence, the binary representation of each integer has a binary prefix of length logarithmic in the length of \( w \) that is followed by a suffix of length \( p \) consisting of 0s only). We call this codification close-to-unary.

**Close-to-unary knapsack**

Given: a sequence \( (1^{w_1}, 1^{p_1}), \ldots, (1^{w_n}, 1^{p_n}) \) of pairs and a pair \( (1^w, 1^p) \), representing the integers \( w_j \ast 2^{p_j}, 1 \leq j \leq n, \) and \( w \ast 2^p \),

Problem: is there a sequence of 0–1 valued variables \( x_1, x_2, \ldots, x_n \) such that \( w \ast 2^p = \sum_{j=1}^{n} x_j \ast (w_j \ast 2^{p_j})? \)
Theorem 4.1 Close-to-unary knapsack is NL-complete.

Proof. For the containment in NL, consider the following algorithm, in which the integers are guessed and summed up (in binary) according to their p-values and compared bitwise with the goal.

Let \( \text{bin}(w \cdot 2^p) = b_1 b_2 \cdots b_m, b_i \in \{0,1\} \), be the binary representation of the goal integer, and \( \text{bin}(w) = a_1 \cdots a_{m-p}, a_i \in \{0,1\} \) the binary representation of \( w \). Let \( B \) be an array of size \( m \) containing \( \text{bin}(w \cdot 2^p) \) such that for all \( 1 \leq i \leq m \) it holds \( B[i] = \begin{cases} 0, & \text{if } i > m - p, \text{ and} \\ a_i, & \text{if } i \leq m - p. \end{cases} \)

Algorithm.

\( \text{Input: } (w_1, p_1), \ldots, (w_n, p_n) \) and the array \( B \)

\( k := m; s := 0 \)

\For {i = 0} {p_{\text{max}}} \text{ do } /* p_{\text{max}} = \max(p_i \mid 1 \leq i \leq n) */

\begin{itemize}
  \item guess (scanning input from left to right)
  \item integers \( (w_{j_1}, p_{j_1}), \ldots, (w_{j_l}, p_{j_l}), l \geq 0 \), such that \( p_{j_1} = \cdots = p_{j_l} = i \),
  \item and compute their sum \( s_i := \Sigma_{j=1}^l w_{j}, \text{ in binary} ; \)
  \item if \( (s \mod 2) \neq (B[k] \mod 2) \) then reject else \( s := s \div 2; k := k - 1; \)
\end{itemize}

if \( s = B[1] \cdots B[k] \) then accept else reject;
end.

It is not hard to see that the algorithm can be executed with non-deterministic logarithmic space. Since the \( w_i \) are given in unary, any of the intermediate sums \( s, s_i \) is polynomially bounded in the size of the input and can be presented on the work tapes. The array \( B \) can be simulated on the work tapes by representing \( w \) and \( p \) in binary. All the operations are computations modulo 2, division by 2 or simple bit comparisons for integers on the work tapes, and computable with logspace.

For hardness, we reduce from the following NL-complete variant of the graph accessibility problem [Jo 75]: given a digraph \( G = (V, E) \) with \( V = \{0,1, \ldots, n\} \), and \( E = E' \cup \{(n,n)\} \) for some \( E \subseteq \{(i,j) \mid 0 \leq i \leq n, 1 \leq j \leq n\} \), the problem consists in deciding whether there is a path of length exactly \( n \) from 0 to \( n \) in \( G \). (\( G \) is such that there are no ingoing edges for node 0 in \( E \) and a loop in node \( n \).)
We construct the close-to-unary knapsack instance corresponding to $G$ as follows. Let $M$ be a value greater than the length of the binary representation of the sum of all vertex numbers; e.g., set $M := 2 \cdot \lceil \log(n+1) \rceil$. Note that $2^cM$ for a constant $c$ is polynomially bounded in $n$. The goal integer is defined with
\[(1^w, 1^p) := 2^M \cdot 2^{2nM} = 2^{(2n+1)M}.\]
For each edge $(i, j) \in E$, $1 \leq i, j \leq n$, there is a sequence of integers
\[(1^{w_{ij}}, 1^{p_{ij}}) := ((2^M - j) \cdot 2^{2M} - (2^M - i)) \cdot 2^{2lM} = (2^M - j) \cdot 2^{2(i+1)M} - (2^M - i) \cdot 2^{2lM} \quad \text{for } 0 \leq l \leq n - 1.\]
Additionally, there are the two integers
\[
(1^{w_0}, 1^{p_0}) := 2^M \cdot 2^0 = 2^M, \quad \text{and} \quad (1^{w_n}, 1^{p_n}) := n \cdot 2^{2M} \cdot 2^{2(n-1)M} = n \cdot 2^{2nM}.
\]
We claim that there is a solution of the close-to-unary knapsack instance defined above iff there is a path of length $n$ from 0 to $n$ in $G$. Assume first that there exists a path of the required form in $G$:
\[(0 = i_0, i_1, (i_1, i_2), \ldots, (i_j, i_{j+1}), \ldots, (i_{n-1}, i_n = n)).\]
Then one checks easily that we obtain a solution of the knapsack with the integers $(1^{w_0}, 1^{p_0}), (1^{w_n}, 1^{p_n})$, and $(1^{w_{i+l}, 1^{p_{i+l}}})$ for $0 \leq l \leq n - 1$.

Conversely, assume that $S$ is an arbitrary solution of the knapsack, i.e., $S$ is the subset of all pairs $(1^{w_i}, 1^{p_i})$ for which $x_i = 1$, $1 \leq i \leq n$. Consider the binary representation of the goal integer that is a 1 followed by a 0-string of length $(2n + 1)M$, and divide the 0-string into blocks $B_{2n+1} \cdots B_0$ of size $M$ each. Then $S$ must be such that it contains
1. the integer $(1^{w_0}, 1^{p_0})$ and exactly one further integer $(1^{w_{i+1}, 1^{p_{i+1}}})$ with $p$-value 0, because otherwise the blocks $B_0, B_1$ could not be cleared;
2. the integer $(1^{w_n}, 1^{p_n})$ and exactly one further integer $(1^{w_{n-1}, 1^{p_{n-1}}})$ with $p$-value $2(n-1)M$ to clear block $B_{2n+1}$; and
3. for each $l$, $0 \leq l \leq n - 1$ exactly one integer with $p$-value $2lM$ such that $\exists j : (1^{w_j}, 1^{p_j}) \in S \iff \exists k : (1^{w_k}, 1^{p_k}) \in S$, to clear the blocks $B_{2(l+1)}, B_{2(l+2)}$.

It is not hard to see that 1.-3. can only be fulfilled if there is a path of length $n$ from 0 to $n$ in $G$. □
Acknowledgement

The author is grateful to Klaus-Jörn Lange for discussions about Theorem 3.1, to Ricard Gavaldà and Kim Gabarró for helpful comments on an earlier version of this report, and especially to Jacobo Torán for many invaluable comments (including the misleading ones).

References


List of research reports (1993).


LSI-93-6-R. “Multilevel use of coherence for complex radiosity environments”, Josep Vilaplana and Xavier Pueyo.

LSI-93-7-R. “A characterization of $\mathbb{P}^{\mathbb{NP}||} = \mathbb{P}^{\mathbb{NP}[\log]}$, Antoni Lozano.


LSI-93-10-R. “Parallel approximation schemes for problems on planar graphs”, Josep Díaz, María J. Serna, and Jacobo Torán.


LSI-93-14-R. “A heuristic search approach to reduction of connections for multiple-bus organization”, Patricia Ávila.


LSI-93-16-R. “The Odissea approach to the design of information systems from deductive conceptual models”, Maria Ribera Sancho and Antoni Olivé.
LSI-93-17-R "Constructing face octrees from voxel-based volume representations", Robert Juan i Ariño and Jaume Solé i Bosquet.

LSI-93-18-R "Discontinuity and pied-piping in categorial grammar", Glyn Morrill.


LSI-93-21-R "Extending a single resolution system towards a distributed society", Karmelo Urzelai.


LSI-93-23-R "Especificació d'una biblioteca de tipus" (written in Catalan), Xavier Franch.

LSI-93-24-R "Proceedings of the Fourth Barcelona-Ulm Workshop on Probabilistic Complexity Classes and Nonuniform Computational Models" (Barcelona, September 13th-17th, 1993), José L. Balcázar and Antoni Lozano (editors).


LSI-93-26-R "Modelo para el control de calidad en LESD basado en la medición del software" (written in Spanish), O. Slávkova.


LSI-93-28-R "Lexicografía computacional: Adquisición automática de Conocimiento Léxico" (written in Spanish), Irene Castellón Masalles.

LSI-93-29-R "Anàlisi de les definicions verbals del diccionari Vox" (written in Catalan), Mariona Taulé Delor.

LSI-93-30-R "The structure of a logarithmic advice class", Montserrat Hermo.

LSI-93-31-R "Toward a realistic semantics of possible worlds for logics of belief", Gustavo Núñez, Matías Alvarado, and Ton Sales.

LSI-93-32-R "Conocimiento en mundos posibles mediante una relación de posibilidad constructiva" (written in Spanish), Matías Alvarado.

LSI-93-34-R “Concatenation versus addition in knapsack problems”, Birgit Jenner.

Internal reports can be ordered from:

Nuria Sánchez
Departament de Llenguatges i Sistemes Informàtics (U.P.C.)
Pau Gargallo 5
08028 Barcelona, Spain
secrelsi@lsi.upc.es