Reversing a Finite Sequence

Loïc Pottier
INRIA Sophia Antipolis

March 6, 1995

[summary by Philippe Dumas]

The concept of reversing a finite sequence is best introduced by an example. Define a sequence of vectors \( x_i \) by the formula \( x_i = f_i(x_{i-1}) \) for \( i = 1, \ldots, p + 1 \); here \( x_0 \) and the functions \( f_i \)'s are given. More precisely the functions \( f_1, \ldots, f_p \) map \( \mathbb{R}^m \) into \( \mathbb{R}^m \) and the last one \( f_{p+1} \) maps \( \mathbb{R}^m \) into \( \mathbb{R} \). For each \( i \), the variable \( x_i \) is a function of \( x_0 \), \( x_i = g_i(x_0) \). Moreover let us assume that all these functions are differentiable. We want to compute the Jacobian matrix \( J_{g_{p+1}}(x_0) \), which expresses the partial derivatives of \( x_{p+1} \) with respect to the components of \( x_0 \). By the chain rule, this matrix is expressed as a product of matrices,

\[
J_{g_{p+1}}(x_0) = J_{f_{p+1}}(x_p) \times J_{f_p}(x_{p-1}) \times \cdots \times J_{f_1}(x_0).
\]

The matrix \( J_{g_{p+1}}(x_0) \) is a row matrix of type \( 1 \times m \), while the matrices in the product are square matrices of type \( m \times m \) except the leftmost one, which is a row matrix of type \( 1 \times m \). The first idea which comes to mind is the following. We compute \( J_{f_1}(x_0) \) and we store it; next we compute \( x_1 \), the Jacobian matrix \( J_{f_2}(x_1) \) and the product \( J_{g_2}(x_0) = J_{f_1}(x_1) \times J_{f_1}(x_0) \); we store this product, we compute \( x_2 \), the matrix \( J_{f_3}(x_2) \), the product \( J_{f_3}(x_2) \times J_{g_2}(x_0) \) and so on. At each step of the computation, we store a \( m \times m \) matrix. If \( m \) is large (a value of about \( 10^6 \) is possible), this method is not practical. So, we apply another strategy. First compute \( x_p \) and the Jacobian matrix \( J_{f_{p+1}}(x_p) \); we store it; next we compute \( x_{p-1} \) and the Jacobian matrix \( J_{f_p}(x_{p-1}) \); we compute the product \( J_{f_{p+1}}(x_p) \times J_{f_p}(x_{p-1}) \) and we store it, and so on. The gain of storage is evident: each time we store a \( 1 \times m \) matrix in place of a \( m \times m \) matrix. But there is a waste of time because we compute again and again the values \( x_1, \ldots, x_p \). Obviously, we could store these values but the available memory has a limited size.

The problem of reversing the sequence \( x_0, x_1, \ldots, x_p \) may be now formulated. We want an algorithm which provides the values \( x_p, x_{p-1}, \ldots, x_1 \) in this order and costs the minimal amount of time, knowing that each computation \( x_i = f_i(x_{i-1}) \) takes one unit of time and only \( r \) values may be stored at a time. Such an algorithm provides the value \( x_i \) only by computation from the previous value \( x_{i-1} \) or by retrieval from memory. Several authors have addressed this problem. Baur and Strassen [2] used the idea we presented as an introduction to study the complexity of partial derivatives. Abbot and Galligo [1] gave an optimality result in the framework of divide-and-conquer algorithms: for such an algorithm, one chooses an index \( q \) between 1 and \( p \), one deals first with the sequence \( x_q, \ldots, x_p \), and next with the sequence \( x_1, \ldots, x_{q-1} \). Grimm, Pottier and Rostaing-Schmidt [3] considered all the algorithms and showed that algorithms of divide-and-conquer type provide the optimal time of computation \( T \). In practice, it is necessary to find a trade-off between \( r \), the number of registers, and \( T \), the number of computations, hence the important quantity is the product \( rT \). Grimm, Pottier and Rostaing-Schmidt gave a lower bound for the product \( (r + 1)T \), which is rather tight and shows that the product \( rT \) has order \( p \ln^2 p \).
\[ > > S > S \]
\[ P \quad X_6 = (0, 3, 5, 6) \]
\[ R \quad X_5 = (0, 3, 5) \]
\[ > S \]
\[ P \quad X_4 = (0, 2, 3, 4) \]
\[ R \quad X_3 = (0, 2, 3) \]
\[ R \quad X_2 = (0, 2) \]
\[ P \quad X_1 = (0, 1) \]

**Figure 1.** The diagram shows the reversing of a sequence \( x_0, \ldots, x_6 \) with 3 registers. Column \( j \) corresponds to term \( x_j \) for \( j = 1, \ldots, 6 \). The symbol \( > \) means that the value is computed but not stored; the symbol \( S \) means that the value is computed and stored; \( P \) means the value is calculated, printed and then thrown away; \( R \) means the value is retrieved from memory, printed and then thrown away. The list \( X_j \) gives the indices \( k \) of the values \( x_k \) which are stored just before term \( x_j \) is printed. The total number of symbols \( >, S \) or \( P \) provides the time of computation.

1. **Reduction to divide-and-conquer algorithms**

The search for an optimal algorithm needs a careful definition of what is an algorithm in this context. The following definition is proposed.

**Definition 1.** A reversal table of the sequence \( x_0, \ldots, x_p \) with \( r \) registers is a family \( (X_{i,j}) \), \( 0 \leq i \leq r_j, 0 \leq j \leq p \), such that

- \( X_{i,j} < X_{i+1,j} \) for \( 0 \leq i < r_j, 0 \leq j \leq p \);
- \( X_{0,j} = 0 \) and \( X_{r_j,j} = j \) for \( 0 \leq j \leq p \);
- \( r_j \leq r \) for \( 0 \leq j \leq p \).

The definition must be understood in the following manner. The list \( X_j = (X_{i,j})_{0 \leq i \leq r_j} \) provides the values \( x_k \) which are stored just before the value \( x_j \) is printed. More precisely the list contains the indices \( k \) arranged in increasing order. See Figure 1 for an example. Notice that the value \( x_0 \) is stored for free because the register used is not taken into account; in fact there are \( r + 1 \) registers used.

To each reversal table \( X = (X_{i,j}) \) is associated its time of computation

\[ t_X = \sum_{0 \leq i \leq r_j \atop 0 \leq j \leq p} t_{i,j}, \quad \text{with} \quad t_{i,j} = X_{i,j} - Y_{i,j}, \]

where \( Y_{i,j} \) is the maximal index of stored values less than \( X_{i,j} \). Line 5 of Figure 1 provides the following values: \( t_{4,2} = 2 \) because the value \( x_2 \) may be obtained at this time only from \( x_6 \); \( t_{4,3} = 0 \) because \( x_3 \) is available from memory, and \( t_{4,4} = 1 \) because \( x_4 \) must be computed from \( x_3 \). The goal is to find a reversal table \( X \) which provides the minimal time of computation \( t_X \). The main theorem is stated as follows.

**Theorem 1.** There exists an optimal reversal table which is of divide-and-conquer type.

We say that a reversal table \( (X_{i,j}) \) is of divide-and-conquer type if there exists an index \( q \) such that

\[ X_{1,p} = X_{1,p-1} = \cdots = X_{1,q} = q. \]

This means that the algorithm computes the value \( x_q \), handles the sequence \( x_q, x_{q+1}, \ldots, x_p \), and next the sequence \( x_0, \ldots, x_{q-1} \).
Figure 2. The diagram shows a divide-and-conquer optimal reversing of a sequence of length $p = 10$ using $r = 3$ registers. The time of computation is $T = 18$.

2. Optimal time

The previous result reduces the search for an optimal algorithm to the consideration of algorithms of divide-and-conquer type.

Theorem 2. The time of any optimal reversal table of a sequence $x_0, \ldots, x_p$ with $r$ registers is given by

$$T(p, r) = k(p + 1) - \binom{r + k}{r + 1},$$

where $k$ is any integer which satisfies

$$\binom{r + k - 1}{r} - 1 \leq p \leq \binom{r + k}{r} - 1.$$

Moreover a reversal table of divide-and-conquer type is optimal if and only if its index $q$ satisfies

$$\binom{r + k - 2}{r} \leq q \leq \binom{r + k - 1}{r}, \quad \text{and} \quad \binom{r + k - 1}{r - 1} - 1 \leq p - q \leq \binom{r + k - 1}{r - 1} - 1.$$

The first part of the assertion appears in [1]. The proof uses an auxiliary function $m_{r,s}$; this function gives the maximal length of a sequence which can be inverted using only $r$ registers and computing only $s$ times each value $x_k$ in the worst case. The proof of the second part relies on the consideration of

$$f(q) = q + T(q - 1, r) + T(p - q, r - 1).$$

This function of the real variable $q$ achieves its minimum on the interval given in the theorem and this minimum is $T(p, r)$. This gives a functional equation for $T(p, r)$, which translates exactly the divide-and-conquer strategy.

It must be noted that for a divide-and-conquer optimal reversal table the number $r$ of registers is exactly the maximal number of times a term of the sequence is computed. One can observe this phenomenon in the example of Figure 2, where the terms $x_1, \ldots, x_{10}$ are respectively computed 3, 2, 2, 2, 1, 1, 2, 2, 1, 2, 1 times.
3. Space-time trade-off

Up to now the number $r$ of available registers was fixed. But it is natural to make the computation more efficient by choosing $r$ as a function of $p$. In this context the quantity of interest is the product $rT(p, r)$.

**Theorem 3.** The product $(r + 1)T$ is greater than a quantity which is equivalent to $p \ln^2 p \ln^{-2} 4$. There exist arbitrary large $p$'s and $r$'s such that the product $(r + 1)T$ is equivalent to $p \ln^2 p \ln^{-2} 4$.

The idea of the proof is to replace the true quantities using the approximations

$$(r + 1)T(p, r) \approx (p + 1)r(k - 1), \quad \left(\frac{r + k}{r}\right) \approx (r + k)^{r+k}r^{-r}k^{-k}.$$

This gives an $r$ which minimizes the product $(r + 1)T$. The result is illustrated by Figure 3.

![Figure 3](image_url)

**Figure 3.** The product $rT(p, r)$ is close to $C_p = p \ln^2 p \ln^{-2} 4$ for $p$ large. Shown here are the sequences $rT(p, r)/C_p$ for $1 \leq p \leq 30$ and $r = 2, \ldots, 10$.

**Bibliography**


[3] Grimm (J.), Pottier (L.), and Rostaing-Schmidt (N.), – A sharp lower bound on the time-space product for reversing a finite sequence. – Preprint, 1995.