ELSEVIER

Theoretical Computer Science 293 (2003) 189–217

Theoretical Computer Science

www.elsevier.com/locate/tcs

Application of Max-Plus algebra to biological sequence comparisons

J.-P. Comet

LAMI, CNRS UMR 8042, University of Evry, Tour Evry 2, 523 Place des terrasses de l'agora, 91025 Évry Cedex, France

Abstract

The classical algorithms to align two biological sequences (Needleman and Wunsch and Smith and Waterman algorithms) can be seen as a sequence of elementary operations in $(\max, +)$ algebra: each line (viewed as a vector) of the dynamic programming table of the alignment algorithms can be deduced by a $(max, +)$ multiplication of the previous line by a matrix. Taking into account the properties of these matrices there are only a finite number of nonproportional vectors. The use of this algebra allows one to imagine a faster equivalent algorithm. One can construct an automaton and afterwards skim through the sequence databank with this automaton in linear time. Unfortunately, the size of the automaton prevents using this approach for comparing global proteins. However, biologists frequently face the problem of comparing one short string against many others sequences. In that case this automaton version of dynamic programming results in a new algorithm which works faster than the classical algorithm.

Keywords: Biological sequence alignment; Dynamic programming; Max-plus algebra

0. Introduction

When a new DNA or protein sequence is determined, it is generally compared to all known sequences in order to find those that are similar to the query. Several indices for computing similarity exist. The first to be used was the global alignment dissimilarity index given by the Needleman and Wunsch algorithm [19]. A global alignment, however, is not necessarily the best representation of biological relationships because some biological functions are associated with patterns or domains. Smith and Waterman [21,22] then generalized the previous algorithm to search for the best local

E-mail address: comet@lami.univ-evry.fr (J.-P. Comet).

0304-3975/03/$ - see front matter
PII: S0304-3975(02)00237-2

alignment. The limitation of this algorithm is obviously the time. Its time complexity is quadratic-proportional to the product of the sequence lengths.

Some heuristics have been devised to overcome this limitation. The algorithm BLAST [1] aims to find the best **non-gapped** alignments. It is very fast, but does not allow gaps. The new BLAST [2] takes gaps into account. Another heuristic method called FASTA [15] first finds local alignments without gaps, and afterwards improves the results by computing the Smith and Waterman alignment in a neighborhood of the non-gapped alignments. Nevertheless, the most rigorous method to align two biological sequences with gaps remains the Smith and Waterman algorithm [22]. In this algorithm only additions and maximizations are needed, one can formulate it in the $(\max,+)$ algebra.

In the first section the $(\max,+)$ algebra is described and relationships between $(max,+)$ matrices and graphs are illustrated. Afterwards, in Section 2, the alignment algorithms using dynamic programming are detailed: ideas of Needleman and Wunsch algorithms are presented in the case of a linear function for gap penalty and then in the case of an affine function. The local alignment algorithm of Smith and Waterman is then presented. Section 3 describes how $(\max,+)$ algebra can be helpful for the dynamic programming algorithms. For the Needleman and Wunsch algorithm with a linear function of gap penalty, when one compares two sequences $A[1,l_A]$ and $B[1,l_B]$ of respective length l_A and l_B, one repeats l_A times the $(\max,+)$-multiplication of a row vector X_l of size l_B+1 by an upper triangular matrix:

$$X_l = X_{l-1} \otimes E_l.$$

The set of matrices $\{E_l\}_l$ is simple: there are as many matrices as letters in the alphabet Σ (4 when comparing DNA sequences and 20 when comparing proteins). Each matrix depends only on the sequence $B[1,l_B]$. The semigroup generated by the set $\{E_a,\ a\in\Sigma\}$ is projectively finite. Then the set of consecutive products of matrices E_l can be pre-computed and an automaton can be built to implement the dynamic programming alignment problem. Section 4 focuses on the well known automaton of Cayley and on the orbit automaton which can be built in this application taking into account the properties of such matrices. When comparing one sequence $B[1,l_B]$ with all sequences of a databank, the automaton is built only once. The algorithm computing the alignment score after having built automata reduces to a simple scan of the sequence $A[1,l_A]$ with the automaton.

Sections 5–8 report different implications of such automata in different cases: Needleman and Wunsch, best occurrence of word, Smith and Waterman and affine gap penalty function case. The last section gives results obtained in a large scale application: the simple case of pattern-matching with errors.

1. Notation, (max, +) algebra and graphs

Our purpose is not to study the $(max,+)$ algebra. However this section allows the reader to become familiar with this formalism, and to summarize some main results useful for the remainder of the paper.

The $(max,+)$ algebra is a traditional name for the semiring $(\mathbb{R}\cup\{-\infty\},max,+)$ denoted by $\mathbb{R}_{\max}$. The two internal operations are noted $\oplus$ and $\otimes$, zero and unit are denoted by ε and e: $\oplus = max$, $\otimes = +$, $\varepsilon = -\infty$ and $e = 0$. For any $\lambda \in \mathbb{R}_{\max}\backslash\{\varepsilon\}$, λ^{-1} is the $(max,+)$ notation for $-\lambda$. This is an example of an idempotent semiring: the first internal operation satisfies $\lambda\oplus\lambda = \lambda$, $\forall\lambda\in\mathbb{R}_{\max}$. Such semirings are known as dioid [3]. This structure is widely used in application to graph theory and operation research [12], and in the study of Discrete Event Dynamic Systems [3,9].

The matrix operations induced by the semiring structure are defined and written as usual. For matrices A and B of the same dimension the addition $A\oplus B$ denotes the matrix where each element is: $(A\oplus B)_{i,j} = A_{i,j}\oplus B_{i,j} = max(A_{i,j},B_{i,j})$. If $C\in\mathbb{R}_{\max}^{n\times p}$ and if $D\in\mathbb{R}_{\max}^{p\times m}$, the product $C\otimes D$ is the matrix in $\mathbb{R}_{\max}^{n\times m}$ defined by

$$(C\otimes D)_{i,j} = \oplus_k C_{i,k}D_{k,j} = \max_k (C_{i,k} + D_{k,j}).$$

We will abbreviate the $(max,+)$ multiplication $C\otimes D$ to CD as usual.

1.1. $(max,+)$ matrices and graphs

From each matrix A in $\mathbb{R}_{\max}^{n\times n}$ a directed weighted graph can be built over n vertices. The vertices are numbered from 1 to n, and edges are defined as follows:

- the edge from i to j exists iff $A_{i,j}\neq\varepsilon$,
- the weight of this edge is $A_{i,j}$.

The weight of a path $(i_1,i_2,\ldots,i_k)$ from i to j, i.e. $i_1 = i$ and $i_k = j$ is the sum of the weights: $A_{i_1i_2}+A_{i_2i_3}+\cdots+A_{i_{k-1}i_k}$. The maximum weight of paths from i to j is defined as the upper bound of these quantities when the path runs over the set of all possible paths from i to j. If no path joins i to j, then this maximum weight is $-\infty$. (When one path exists from i to j going through a vertex belonging to a circuit of positive weight, then the maximum weight is $+\infty$.) One can prove that the upper bound of weight from i to j is

$$(A\oplus A^2\oplus A^3\oplus\cdots\oplus A^n\oplus\cdots)_{i,j}.$$

Let us denote by * and $^+$ the two following operation:

$$A^* = I\oplus A\oplus A^2\oplus\cdots\oplus A^n\oplus\cdots$$

$$A^+ = A\oplus A^2\oplus\cdots\oplus A^n\oplus\cdots = AA^*,$$

where I is the identity matrix for $\mathbb{R}_{\max}^{n\times n}$. Then the upper bound of weight of any path from i to j is given by the component (i,j) of the matrix A^+.

1.2. Resolution of linear equations: $Ax\oplus b = x$

The Needleman and Wunsch algorithm can be seen as a problem of maximum cost in a weighted directed graph [16]. The previous paragraph says the $(max,+)$ algebra is

a good structure for such a problem. To write such an algorithm in $(max,+)$ algebra requires knowledge of linear system resolution. The following theorem and property will be useful in the following sections. Proofs of these results can be found in [3].

Theorem. *If there are only negative or null circuits in the graph associated with matrix A, the equation $Ax \oplus b = x$ has a solution that is given by $x = A^* b$. If in addition there is no circuit with null weight, the solution is unique.*

Property. *If the graph associated with the matrix A has no circuit with positive weight, then $A^* = I \oplus A \oplus A^2 \oplus A^3 \oplus \cdots \oplus A^{n-1}$ where n is the dimension of matrix A.*

2. Dynamic programming alignments

Before using the $(max,+)$ algebra, let us first detail the two algorithms of global and local alignments: the Needleman and Wunsch algorithm and Smith and Waterman algorithm. These two algorithms are based on the dynamic programming method.

2.1. Global alignment: Needleman and Wunsch algorithm

Needleman and Wunsch were the first to use dynamic programming to compare two biological sequences [19]. Their algorithm finds a global alignment between two sequences of any length. Let A and B be two sequences of respective length l_A and l_B. This alignment is obtained by maximization of a cost called the "*edit cost*". It tries to transform the first sequence A into the second one B with three exclusive basic operations:

- Mutation: the letter $A[i]$ in sequence A becomes the letter $B[j]$ in B with a cost of $\sigma(A[i], B[j])$ independent of positions i and j. The *substitution* costs $\sigma(A[i], B[j])$ are given by a matrix called the *substitution matrix*. We will abbreviate this notation to $\sigma_{i,j}$.
- Insertion: the letter $B[j]$ is inserted in the sequence B with a gap cost δ ($\delta \geqslant 0$).
- Deletion: the letter $A[i]$ in sequence A is deleted with the same gap cost δ.

Each previous operation can be seen as a particular alignment pair: respectively $\left[\begin{smallmatrix} A[i] \\ B[j] \end{smallmatrix}\right]$, $\left[\begin{smallmatrix} - \\ B[j] \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} A[i] \\ - \end{smallmatrix}\right]$. The main recurrence is then

$$N(i,j) = \max \begin{pmatrix} N(i-1,j-1) + \sigma_{i,j}, \\ N(i-1,j) - \delta, \\ N(i,j-1) - \delta \end{pmatrix}, \tag{1}$$

where $N(i,j)$ corresponds to the alignment score between prefixes $A[1,i]$ and $B[1,j]$.

Such a gap penalty schema does not represent what we observe in practice. One insertion event of k letters is more often observed than k distinct insertion events of

one letter. The first algorithm has been improved to take into account different schemas of gap penalty [13,20]. If the penalty of an insertion/deletion (indel) of k consecutive letters is chosen as an affine function: $g(k) = go + ge(k-1)$ with $go \geqslant ge > 0$, where go means gap-open penalty and ge gap-extend penalty, then the basic recurrence becomes

$$N(i,j) = \max \begin{pmatrix} N(i-1,j-1) + \sigma_{i,j}, \\ -go + \max\limits_{1 \leqslant k < i} (N(i-k,j) - (k-1)ge), \\ -go + \max\limits_{1 \leqslant k < j} (N(i,j-k) - (k-1)ge) \end{pmatrix}. \tag{2}$$

Initial conditions are given by

$$N(0,0) = 0$$

$$N(i,0) = -g(i), \quad 1 \leqslant i \leqslant l_A,$$

$$N(0,j) = -g(j), \quad 1 \leqslant j \leqslant l_B.$$

The final Needleman and Wunsch score (NW score) is the value $N(l_A, l_B)$.

2.2. Local alignment: Smith and Waterman algorithm

Smith and Waterman generalized the previous algorithm to search for the best local alignment. The local alignment is a better representation of biological relationships because some biological functions are associated with patterns or domains. It corresponds to the NW alignment of the subsequences that maximize the NW score. Given a substitution matrix σ and the affine gap cost function $g(.)$ the Smith and Waterman algorithm computes efficiently the alignment that maximizes the alignment score. Its recurrence is given by

$$S(i,j) = \max \begin{pmatrix} 0, S(i-1,j-1) + \sigma_{i,j}, \\ -go + \max\limits_{1 \leqslant k < i} (S(i-k,j) - (k-1)ge), \\ -go + \max\limits_{1 \leqslant k < j} (S(i,j-k) - (k-1)ge) \end{pmatrix}.$$

Initializations are given by

$$S(0,0) = 0,$$

$$S(i,0) = 0, \quad 1 \leqslant i \leqslant l_A,$$

$$S(0,j) = 0, \quad 1 \leqslant j \leqslant l_B.$$

Let $SW(A,B)$ be the maximal value in the matrix S. This value is the Smith and Waterman score (SW score). Of course if the penalty function is linear, the recurrence

is similar:

$$S(i,j) = \max \begin{pmatrix} 0, S(i-1,j-1) + \sigma_{i,j}, \\ S(i-1,j) - \delta, \\ S(i,j-1) - \delta \end{pmatrix}. \tag{3}$$

Subsequent improvements and extensions have been made by Waterman and Eggert [25] to allow identification of all non-intersecting similar subsequences with similarity score at or above a preset level. Miller and Myers [16] have applied the Hirschberg principle "*divide and conquer*" to reduce the memory space necessary for computing the alignment, and in another paper [17] they have optimized the alignment algorithm in the case of a concave function for gap penalties. Here we focus only on basic recurrences of dynamic programming.

3. Dynamic programming alignments and (max, +) algebra

3.1. Method of transfer matrices for computing the NW score

For two sequences A and B of respective length l_A and l_B, the classical dynamic programming alignment is totally symmetrical. In practice, however, one is often interested in comparing a new sequence against all known sequences in a databank. In this case we make a distinction between the sequences. The new sequence will be called the "*query sequence*" while all databank sequences are called "*target sequences*". In the case of comparing A and B, let us choose B as the "*query sequence*". In the comparison of B to a databank, all computations on B not dependent on the target sequences, can be done once beforehand in a preprocessing step.

First, let us consider the simpler algorithm with a linear function for gaps: $g(k) = -k \times \delta$. With the $(max, +)$ notations the recurrence 1 becomes

$$N(i,j) = \sigma_{i,j} N(i-1,j-1) \oplus \delta^{-1} N(i-1,j) \oplus \delta^{-1} N(i,j-1). \tag{4}$$

The initial conditions are given by

$$\begin{aligned} &N(0,0) = 0 = e, \\ &N(i,0) = g(i) = \delta^{-i} \quad \forall i \in [1, l_A], \\ &N(0,j) = g(j) = \delta^{-j} \quad \forall j \in [1, l_B]. \end{aligned} \tag{5}$$

Let X_n be the row vector of size $l_B + 1$ corresponding to the nth row of the usual dynamic programming table of the NW algorithm,

$$X_n = (N(n,0)\ N(n,1)\ N(n,2)\ \cdots\ N(n,l_B)). \tag{6}$$

One can write the recurrence with the vectors X_n:

$$X_n = X_{n-1} C_n \oplus X_n D_n \tag{7}$$

with

$$C_n = \begin{pmatrix} \delta^{-1} & \sigma_{1,n} & . & \dots & . \\ . & \delta^{-1} & \sigma_{2,n} & \dots & . \\ . & . & \delta^{-1} & \ddots & . \\ \vdots & \vdots & \vdots & \ddots & \sigma_{l_B,n} \\ . & . & . & . & \delta^{-1} \end{pmatrix}, \quad D_n = \begin{pmatrix} . & \delta^{-1} & . & . & . \\ . & . & \delta^{-1} & . & . \\ . & . & . & \ddots & . \\ . & . & . & . & \delta^{-1} \\ . & . & . & . & . \end{pmatrix},$$

where

- $\sigma_{i,n}$ is the value of the substitution matrix for letters $B[i]$ and $A[n]$,
- the unspecified values are equal to $-\infty$.

All matrices D_n are independent of n; so one denotes them simply by D. Since all non-null elements of D are in the first over-diagonal, the associated graph has no circuit. From the property of Section 1.2 the sequence $(\bigoplus_{i=0}^{k} D^i)_{k\in\mathbb{N}}$ converges. More precisely all powers of $D, (D^i)_{i\geqslant l_B}$ are null. We have

$$D^* = \bigoplus_{i=0}^{\infty} D^i = \begin{pmatrix} e & \delta^{-1} & \delta^{-2} & \delta^{-3} & \dots & \delta^{-l_B} \\ . & e & \delta^{-1} & \delta^{-2} & \dots & \delta^{-l_B+1} \\ . & . & e & \delta^{-1} & \dots & \delta^{-l_B+2} \\ . & . & . & \ddots & \ddots & \vdots \\ . & . & . & . & e & \delta^{-1} \\ . & . & . & . & . & e \end{pmatrix}. \tag{8}$$

Whatever the row vector b, bD^* is a solution of the linear equation $X = XD \oplus b$. In particular we have

$$X_n = X_n D \oplus X_{n-1} C_n = X_{n-1} \underbrace{C_n D^*}_{E_n}, \quad X_n = X_{n-1} E_n.$$

E_n is upper triangular:

$$\text{if } i < j,\ E_{i,j} = (\delta^{-2} \oplus \sigma_{i,n}) \otimes \delta^{-(j-i-1)} = \max(-2\delta, \sigma_{i,n}) - (j-i-1)\delta,$$

$$\text{if } i > j,\ E_{i,j} = \varepsilon = -\infty,$$

$$\text{if } i = j,\ E_{i,j} = \delta^{-1} = -\delta.$$

One can derive the following result.

Theorem 1. *Let* $X_0 = (e, \delta^{-1}, \delta^{-2}, \ldots, \delta^{-l_B})$. *The Needleman and Wunsch score between sequences A and B is equal to the last component of vector*:

$$X_0 E_1 E_2 \cdots E_{l_A}.$$

Proof. X_0 is equal to the first row of the usual dynamic programming algorithm (cf. Eq. (5) and (6)). Moreover we just proved that $X_n = X_{n-1} E_n$. Each matrix product corresponds to the passing from one row to the following in the usual dynamic programming table of NW algorithm.

Since the NW score is read at the position (l_A, l_B) of the usual dynamic programming table, it is equal to the last component of the last vector. □

The transfer matrices are simpler under the additional constraint: $\delta^{-2} \leqslant \sigma_{\min}$ where $\sigma_{\min}$ is the minimal value of the substitution matrix. This constraint can be interpreted as follows. When aligning two sequences, one substitution will be preferred to two consecutive indels. This corresponds to the minimum number of mutational events.

Under this assumption the transfer matrices have similar shape but the terms become:

$$\text{if } i < j,\ E_{i,j} = \sigma_{i,n} \otimes \delta^{-(j-i-1)} = \sigma_{i,n} - (j - i - 1)\delta,$$

$$\text{if } i > j,\ E_{i,j} = \varepsilon = -\infty,$$

$$\text{if } i = j,\ E_{i,j} = \delta^{-1} = -\delta.$$

3.2. The semigroup of transfer matrices E_n

For a given sequence B, 4 or 20 matrices E_n exist according to the type of sequence: if it is a DNA sequence or a RNA sequence, 4 matrices will be necessary. If the sequence is a protein, 20 matrices are required. The transfer matrix E_i corresponds to letter $A[i]$. So the generators are simply $\{E_a,\ a \in \Sigma\}$ where Σ is the alphabet used.

If one can describe the structure of the set of matrices obtained by consecutive products of transfer matrices, one might accelerate the scanning process. First let us show that for a given sequence B, the set of matrices which can be obtained by such products is *projectively finite*. Let us define the *projective space*.

Definition 2. We define the $(n-1)$-dimensional $(max, +)$ projective space, denoted by $\mathbb{P}\mathbb{R}^n_{\max}$, as the quotient of $\mathbb{R}^n_{\max}$ by the parallelism relation

$$\text{if } u \in \mathbb{R}^n_{\max} \text{ and } v \in \mathbb{R}^n_{\max} \quad u \sim v \ \Leftrightarrow\ \exists \lambda \in \mathbb{R}_{\max} \backslash \{\varepsilon\}, \quad u = \lambda v.$$

The projective space $\mathbb{P}\mathbb{R}^{n\times n}_{\max}$ is defined similarly as the quotient of the space of $(\max, +)$ matrices $\mathbb{R}^{n\times n}_{\max}$ by the parallelism relation:

$$\text{if } U \in \mathbb{R}^{n\times n}_{\max} \text{ and } V \in \mathbb{R}^{n\times n}_{\max} \quad U \sim V \ \Leftrightarrow\ \exists \lambda \in \mathbb{R}_{\max} \backslash \{\varepsilon\}, \quad U = \lambda V.$$

We say that a subset $S \subset \mathbb{R}^{n\times n}_{\max}$ is ***projectively finite*** if the quotient set $S/\sim$ is finite, i.e. iff there are only finitely many pairwise non-proportional elements in S.

We denote by $\langle E_a,\ a\in\Sigma\rangle$ the $(\max,+)$ multiplicative semigroup generated by $\{E_a, a\in\Sigma\}$. It is the set of matrices which can be written as $E_{a_1}E_{a_2}\cdots E_{a_k}$, with $a_1a_2\cdots a_k \in \Sigma^*$.

Theorem 3. *The semigroup* $\langle E_a,\ a\in\Sigma\rangle$ *is* projectively finite.

Proof. Let us define $\tilde{E}_a = \delta E_a,\ \forall a\in\Sigma$.

Let us build the following automaton:

- The set of states is the set of ordinates of any row vector $\mathscr{S} = \{1,2,\ldots,l_B,l_B+1\}$.
- For each letter $a\in\Sigma$, such that $(\tilde{E}_a)_{i,j}\neq\varepsilon$ a transition from i to j exists with the label a. The transition will be denoted by $(i \xrightarrow{a} j)$.
- One associates with each transition $(i \xrightarrow{a} j)$ a weight defined as $(\tilde{E}_a)_{i,j}$.

This automaton is a graph with several edges from i to j. Since the matrices are upper triangular, the edges $(i\to j)$ are such that $i\leqslant j$. Each circuit is a loop from k to k with a null weight.

Let M be a matrix in the semigroup $\langle\tilde{E}_a,\ a\in\Sigma\rangle$. M can be written as $M=\bigotimes_{i=1}^{l}(\tilde{E}_{a_i})$. Since each matrix $\tilde{E}_a,\ a\in\Sigma$, has e on the diagonal, the diagonal components of M are equal to e. $M_{i,j}$ represents the maximum of weight of paths with label $a_1a_2a_3\ldots a_l$ from i to j.

Let us consider in the previous automaton any path from i to j. Let $i=i_1,i_2,i_3\cdots i_k=j$ be this path. The sequence $(i_l)_{l=1\ldots k}$ is increasing. There is only a finite set of paths from i to j if one does not count the loops. Since loop weights are null, the weight of any path from i to j is in a finite set. So $M_{i,j}$ belongs to a finite set. □

Theorem 4. *Let* $N(A,B)$ *be the Needleman and Wunsch score between two sequences of length* l_A *and* l_B. *Then for a sequence* B, *the variable*

$$N(A,B)+\delta\times l_A = \delta^{l_A}\otimes N(A,B)$$

runs over a finite set of values when the sequence A *runs over* Σ^*.

Proof. The previous demonstration proves that the semigroup generated by matrices $\{\tilde{E}_a=\delta E_a,\ a\in\Sigma\}$ is finite. Moreover, the consecutive product of matrices $(\bigotimes_{i=1}^{l_A}\tilde{E}_{A[i]})$ can be written as

$$\left(\bigotimes_{i=1}^{l_A}\tilde{E}_{A[i]}\right) = \bigotimes_{i=1}^{l_A}\delta E_{A[i]}$$

$$= \delta^{l_A}\otimes\left(\bigotimes_{i=1}^{l_A}E_{A[i]}\right)$$

The component (l_B+1) of vector $X_0(\bigotimes_{i=1}^{l_A}\tilde{C}_{A[i]})$ is

$$\left(X_0\bigotimes_{i=1}^{l_A}\tilde{E}_{A[i]}\right)_{l_B+1}=\delta^{l_A}\otimes\underbrace{X_0\left(\bigotimes_{i=1}^{l_A}E_{A[i]}\right)_{l_B+1}}_{N(A,B)}$$

Thus the result is proved. □

This result does not depend on the nature of the parameters $\sigma_{i,j}$: these parameters can be natural or real numbers. In both cases the semigroup $\langle E_a,\ a\in\Sigma\rangle$ is projectively finite.

One can give an upper bound of the size of the semigroup $\langle\tilde{E}_a,\ a\in\Sigma\rangle$. Let $\mathcal{N}$ be the set of values not equal to $-\infty$ present at any position in any matrices $\{\tilde{E}_a,\ a\in\Sigma\}$. One upper bound is

$$\mathcal{M}=|\mathcal{N}|^{(n(n-1))/2}(1+|\mathcal{N}|)^{(n(n-1)(n-2))/6}.$$

4. Automata

Each matrix E_a depends on the fixed query sequence $B[1,l_B]$ and its size is $(l_B+1)\times(l_B+1)$. The semigroup $\langle E_a,\ a\in\Sigma\rangle$ is infinite (the diagonal components tend to $-\infty$). But the semigroup $\langle\tilde{E}_a=\delta E_a,\ a\in\Sigma\rangle$ is finite.

We would like to build an automaton which associates with any word w the matrix in this finite semigroup that corresponds to the associated consecutive products. If $w=abc$, the associated matrix with w is given by $E=\tilde{E}_a\tilde{E}_b\tilde{E}_c$. Let us introduce the mapping from words in Σ^* to the semigroup $\langle\tilde{E}_a,\ a\in\Sigma\rangle$ defined by the canonical extension of the following application:

$$\begin{aligned}\phi:\Sigma&\to\mathbb{R}_{\max}^{(l_B+1)\times(l_B+1)},\\ a&\to\tilde{E}_a.\end{aligned}$$

For two words m_1 and m_2 in Σ^+ $\phi(m_1.m_2)=\phi(m_1)\otimes\phi(m_2)$.

4.1. Cayley automata

Cayley automaton has a state for each element of group. A transition exists from one state to a second one when the element from the second state is equal to the product of the element of the first state and one of the generators. The linear representation of the Cayley automaton is called the *regular representation* in the theory of semigroups. However the term Cayley automaton is preferred in this case because of its intuitive graph. More formally the structure of Cayley automaton is the following.

- There is a unique initial state, corresponding to the identity matrix,
- all states are final,
- the transition $\tilde{E}_1\xrightarrow{a}\tilde{E}_2$ exists if $\tilde{E}_1\tilde{E}_a=\tilde{E}_2$, where $\tilde{E}_1$ and $\tilde{E}_2$ are two matrices in the semigroup.

This automaton is finite since the semigroup $\langle \tilde{E}_a,\ a \in \Sigma \rangle$ is finite. It computes $\phi(A)$ for each sequence A in **time linear** in the sequence length. The score $N(A,B)$ can also be deduced from the automaton

$$N(A,B) = \delta^{-l_A} X_0 \phi(A) \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon \\ e \end{pmatrix}.$$

In the next section we will show that this property is generalizable in a certain sense. If ψ is the unique morphism such that $\forall a \in \Sigma,\ \psi(a) = E_a$ and if the set $X_0\psi(A),\ A \in \Sigma^*$, is projectively finite, one can construct another automaton from which the NW score is deducible.

4.2. Orbit automata

We are not interested in the global semigroup $\langle E_a,\ a \in \Sigma \rangle$, but only in the structure of the orbit of the initial state under actions of the elements of the semigroup. In other words we attempt to compute for each word $w \in \Sigma^*,\ X_0\psi(w)$, where the application ψ is the unique morphism such that $\forall a \in \Sigma,\ \psi(a) = E_a$.

The idea is to construct an automaton analog to the Cayley's but on the orbit of the initial state. This method is well known for boolean automata. In the case of weighted automata it is known but less common. This method is due to Choffrut [7] and has been used for automata with multiplicities over the $(max, +)$ semiring (see for example [10,18]).

Let us build the equivalence classes of orbit $O = \{X_0\psi(A),\ A \in \Sigma^*\}$ with the congruence of proportionality: two elements o_1 and o_2 of the orbit O belong to the same class iff there is a $\lambda \in \mathbb{R}_{\max} \backslash \{-\infty\}$ such that $o_1 = \lambda \otimes o_2$. Since the semigroup $\langle E_a,\ a \in \Sigma \rangle$ is projectively finite, the set O is projectively finite too. The set of equivalence classes is therefore finite.

One can build an automaton whose vertices correspond to these equivalence classes, and which compute the score $N(A,B)$ in linear time in the length of the *target sequence*. Generally this automaton is not minimal.

- Each state i of the automaton corresponds to an equivalence class of O. We associate with state i one vector of this equivalence class, denoted by u_i.
- The initial state is associated with the equivalence class of the initial vector $X_0 = (e, \delta^{-1}, \delta^{-2}, \ldots, \delta^{-l_B})$.
- All states are final.
- The transition $i \xrightarrow{a} j$ exists iff there is $\lambda \in \mathbb{R}_{\max}$ such that $u_j = \lambda(u_i E_a)$, i.e. if vectors u_j and $u_i E_a$ are proportional. If $\lambda = e$, both vectors u_j and $(u_i E_a)$ are equal.
- We associate with transition $i \xrightarrow{a} j$ the weight λ, which represents the proportionality coefficient between the vector u_j and the vector $(u_i E_a)$: $u_j = \lambda(u_i E_a)$.

This automaton computes the vector $X_0\psi(A)$ for any sequence $A \in \Sigma^*$. It is a finite state automaton since the set of equivalence classes is finite. The vector $X_0\psi(A) = X_0E_{A_1}E_{A_2}\cdots E_{A_{l_A}} = X_{l_A}$ is computed in **linear time**. Indeed the automaton gives directly the equivalence class of vector X_{l_A}. To have the exact vector, one needs to compute the proportionality coefficient λ between the vector X_{l_A} and the vector associated with the equivalence class. This coefficient can be calculated in linear time during the scanning process:

$$\lambda = \bigotimes_{k=1}^{l_A} \lambda_k,$$

where λ_k is the proportionality coefficient associated with the kth transition of the path in the automaton.

5. Orbit automaton for the Needleman and Wunsch algorithm

The first step is the construction of the automaton. After the automaton has been built, the sequence comparison between the query sequence B and any target sequence A reduces to going through the automaton along a path of label A.

5.1. Construction of orbit automaton: algorithm

One enumerates all words in Σ^* until the orbit automaton is entirely built. This process finishes since the automaton is finite. Any order for enumerating words can be chosen but we implemented the *military order*: the first order is the length of the word and the second is the alphabetic order. For the alphabet $\Sigma = \{a, b\}$, the military order is $\emptyset$, a, b, aa, ab, ba, bb, aaa, aab … where $\emptyset$ is the null string.

We recall that on the alphabet $\Sigma = \{a_1, a_2, \ldots, a_{|\Sigma|}\}$ and for a given word w in Σ^*, any word of $\{wa_i,\ a_i \in \Sigma\}$, will be called a *son* of w.

To build the automaton two lists of states have to be maintained: the list L_1 is the list of all already built states, and L_2 is the list of all states of which we have not yet explored the sons.

Algorithm.

(1) Construction of matrices $E_a,\ a \in \Sigma$.
(2) The unique initial state s_0 is associated with the initial vector $X_0 = (e, \delta^{-1}, \delta^{-2}, \ldots, \delta^{-l_B})$. $u_0 = X_0$.
Initialization of the two lists: $\begin{cases} L_1 = \{s_0\}, \\ L_2 = \{s_0\}. \end{cases}$
(3) For each state s_i (of which the associated vector is u_i) of L_2, one explores the sons.
- For each letter $a \in \Sigma$:
  - Compute $u = u_iE_a$.

- If there is a number $\lambda \in \mathbb{R}_{\max}$ and a state $s_j \in L_1$ such that $u_j = \lambda u$, i.e. if u is proportional to the vector of one of the states of L_1, then create one transition from s_i to s_j, with label a and coefficient λ.
- Else create a new state s_k, which is added to L_1 and to L_2. The vector associated with state s_k is $u_k = u$.

• Remove s_i from list L_2.

(4) End when list L_2 is empty.

This implemented algorithm is one of the most naïve algorithms since it lists all words according to the military order. It can be improved by using some more efficient algorithms for computing finite semigroups [8].

During phase (3) one needs to make the multiplication $u = u_i E_a$. This calculation is very time consuming: its complexity is $O((l_B+1)^2)$. Then we come back to the equation $X = u_i C_a \oplus XD$ (cf. Eq. (7)). The solution is given by the dynamic programming recurrence with linear complexity $O(l_B+1)$.

This automaton recognizes all words of Σ^*. It is deterministic. Fig. 1 presents the orbit automaton in a simple case. Table 1 gives the size of the Cayley automaton and the size of the orbit automaton for different query sequences. Moreover, this table gives the **depth** of the orbit automaton, which is the maximal length of words that have been enumerated (according to the military order), to build the automaton. Because of the chosen order (i.e. military) no other order can build the automaton using only words with length strictly less than the depth. The orbit automaton size is clearly lower than the Cayley's one. Thus we will consider only orbit automata in the remainder of the paper.

5.2. Computing the score

The algorithm skims the target sequence letter by letter. As one goes along the sequence one maintains the $(max,+)$ product of successive transition weights: $p_l = \bigotimes_{k=1}^{l} \lambda_k$ where λ_k is the proportionality coefficient for the kth transition.

Let us suppose that the final state is k. The vector u_k gives the NW score up to a coefficient. The last row of the usual dynamic programming table is obtained by the $(max,+)$ multiplication of the vector u_k by the product of successive transition weights. Then the score NW is equal to the $(max,+)$ product of the last component of vector u_k by the product of successive transition weights.

$$N(A,B) = u_k[l_B+1]\left(\bigotimes_{k=1}^{l_A}\lambda_k\right) = X_0\psi(A)\begin{pmatrix}\varepsilon\\ \vdots\\ \varepsilon\\ e\end{pmatrix}$$

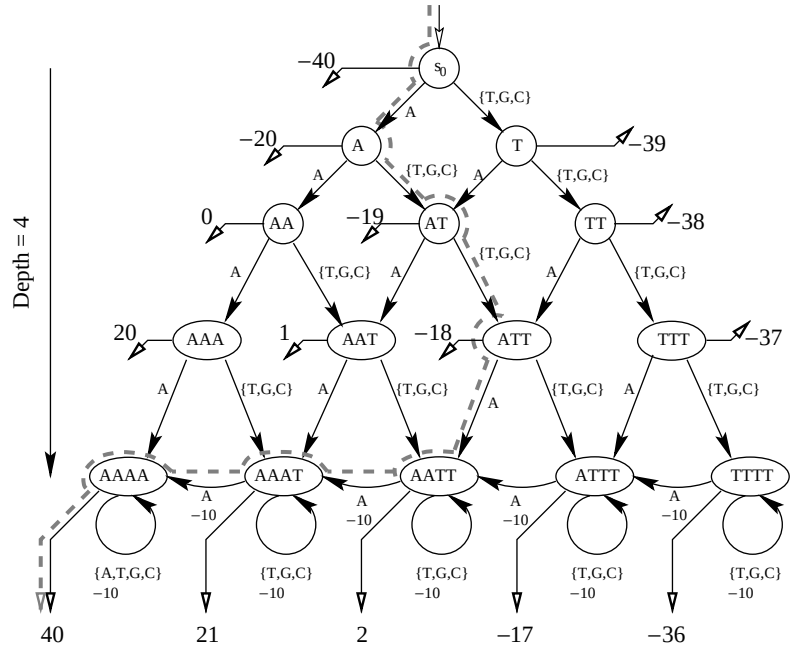

Fig. 1. Orbit automaton: the comparison of a query sequence against a databank with the Needleman and Wunsch algorithm reduces to going through an automaton with the databank sequences. In this example we have chosen for the query sequence the word **AAAA**, and parameters are respectively: $\sigma(a,a)=10\ \ \forall a\in\{A,T,G,C\},\ \ \sigma(a,b)=-9\ \ \forall a,b\in\{A,T,G,C\}$ for $a\neq b$ and $go=ge=-10$. **Scanning a target sequence:** the initial state is s_0. For each letter "a" read in the target sequence, we move in the automaton according to the transition of label "a". At each stage one maintains the $(max,+)$ product of proportionality coefficients (weight associated with transition). Coefficients not shown are 0. All states are final. The score associated with each state (shown by an outgoing arrow) corresponds to the last component of the associated vector. **The final score** is obtained by the $(max,+)$ multiplication of the score associated with the final state by the $(max,+)$ product of proportionality coefficients. For example the path shown by a dotted line corresponds to the scanning of the sequence **ATGAAA**. The final score is $40-10-10=20$. **The structure** of this automaton is due to the intrinsic structure of the query sequence AAAA. Transitions with non-null proportionality coefficient generally can go to a state in a lower depth level (example: from AAAT to AT).

Then we can state the following.

Theorem 5. *Let B be the query sequence. After the construction of the automaton the computation of the Needleman and Wunsch score between B and any sequence A is* linear *in length of sequence A.*

6. Research of the best occurrence of a word

The problem of the best occurrence of a word in a text is related to the previous problem. We compare one short sequence (word) to a long sequence. The

Table 1
Sizes of automata generated by $(\max, +)$ matrices for the Needleman and Wunsch algorithm. Query sequences are DNA sequences. The substitution matrix is $+10$ for diagonal components and -9 elsewhere, $go = ge = -10$

Words	Cayley automaton size	Orbit automaton size	Depth
AAAA	15	15	4
ATTA	84	43	5
ATCG	401	84	6
ATCGA	1571	199	8
ATCGAT	5329	439	9
ATCGATC	15272	919	10
ATCGATCG	39048	1873	12

problem is now to determine the locations of subsequences in the long sequence that are the closest to the word. The first problem of pattern matching is to find all exact occurrences of a pattern in a long text. Such a problem can be efficiently solved by different algorithms [6,14]. Baeza-Yates and Gonnet [4] have given an efficient algorithm of *pattern matching* without error which uses dynamic programming in a degenerate case. In this case a row is coded with 64-bit words and the passing from one row to the following can be done using logic operations on bit-words.

Approximate pattern matching considers three different type of errors: insertions, deletions and mismatches (substitutions), but all mistakes have the same weight. Baeya-Yates and Perleberg developed an algorithm which efficiently finds approximate patterns when the maximum error rate is small [5]. The approach of Tarhio and Ukkonen [23] has been to modify the algorithm of Boyer-Moore [6] to allow errors. A generalization of the Baeza-Yates and Gonnet algorithm led Wu and Manber to create a new algorithm called ***agrep*** (**a**pproximative **grep** [26,27,28]). However, we are looking for subsequences that are the closest to the word in the sense of Needleman and Wunsch, so our method must use the substitution matrix unlike the previous algorithms.

Let B be the word, l_B its length, A the long sequence of length l_A. Dynamic programming solves this problem and the recurrence is the same as for NW scores (cf. Eq. (4)), but the algorithm differs by initializations (compare to Eq. (5)):

$$\begin{aligned} &N(0,0) = 0 = e, \\ &N(i,0) = 0 = e \quad \forall i \in [1, l_A], \\ &N(0,j) = g(j) = \delta^{-j} \quad \forall j \in [1, l_B]. \end{aligned} \tag{9}$$

Now the interesting score is not the last component of the usual dynamic programming table, but the maximum in the last column: the gap penalties in the sequence A after each approximative occurrence of B are null.

With the same notations as the previous section one can write the relation between the two vectors X_n and X_{n-1}: $X_n = X_{n-1}C'_n \oplus X_n D_n$ with

$$C'_n = \left(\begin{array}{c|cccc|c} e & \sigma_{1,n} & . & \dots & . & . \\ \hline . & \delta^{-1} & \sigma_{2,n} & . & . & . \\ . & . & \delta^{-1} & \ddots & . & . \\ . & . & . & \ddots & \sigma_{l_B-1,n} & . \\ . & . & . & . & \delta^{-1} & \sigma_{l_B,n} \\ \hline . & . & . & . & . & e \end{array}\right), \qquad D_n = \begin{pmatrix} . & \delta^{-1} & . & . & . \\ . & . & \delta^{-1} & . & . \\ . & . & . & \ddots & . \\ . & . & . & . & \delta^{-1} \\ . & . & . & . & . \end{pmatrix}.$$

Notice that the matrix C'_n is not the same as the matrix involved in the case of NW score (cf. Eq. (7)). The first and last element of the diagonal are now null. The first one imposes $X_n(0) = 0,\ \forall n \in [1, l_A]$, and the second one prevents weighting the insertions/deletions at the end of the target sequence. The score is then read in the last component of the last vector. Denoting $F_n = C'_n D^*$, one has the following recurrence:

$$X_0 = (e, \delta^{-1}, \delta^{-2}, \dots, \delta^{-l_B}),$$

$$X_n = X_{n-1}F_n.$$

The matrix F_n is computable:

$$F_n = \left(\begin{array}{c|ccccc|c} e & & & F_{1,j} & & & F_{1,l_B+1} \\ \hline \varepsilon & \delta^{-1} & & & & & \\ \varepsilon & \varepsilon & \delta^{-1} & & & F_{i,j} & F_{i,l_B+1} \\ . & . & . & \delta^{-1} & & & \\ . & . & & \varepsilon & \delta^{-1} & & \\ \hline \varepsilon & \varepsilon & \dots & \varepsilon & \varepsilon & e & \end{array}\right) \tag{10}$$

with

$$F_{i,j} = \begin{cases} \text{if} \quad i = j = 1, & e = 0, \\ \text{if} \quad i = j = l_B + 1, & e = 0, \\ \text{if} \quad i = 1 \neq j, & (\delta^{-1} \oplus \sigma_{i,n}) \otimes \delta^{-(j-i-1)} \\ & \quad = \max(-\delta, \sigma_{i,n}) - (j - i - 1)\delta, \\ \text{if} \quad i < j, & (\delta^{-2} \oplus \sigma_{i,n}) \otimes \delta^{-(j-i-1)} \\ & \quad = \max(-2\delta, \sigma_{i,n}) - (j - i - 1)\delta, \\ \text{if} \quad i > j, & \varepsilon = -\infty, \\ \text{if} \quad i = j \begin{pmatrix} i \neq 1, \\ i \neq l_B + 1 \end{pmatrix}, & \delta^{-1} = -\delta. \end{cases}$$

Each matrix $F_a,\ a \in \Sigma$ differs from the corresponding matrix of the NW problem. This difference is crucial for the structure of the semigroup. The most important effect is

clearly that the set $O=\{X_0F_{a_1}F_{a_2}\dots F_{a_l},\ a_1a_2\dots a_l\in\Sigma^*\}$ is finite but the semigroup is not projectively finite any more.

Theorem 6. *Let X_0 be a vector with finite coordinates*: $X_0(i)\neq\varepsilon\ \forall i\in[1,l_B+1]$. *The set $\{X_0F_{a_1}F_{a_2}\cdots F_{a_k},\ a_1a_2\cdots a_k\in\Sigma^*\}$ is finite.*

Proof. Let us first build the following automaton.

- The set of states is the set of coordinates of vectors $\mathscr{S}=\{1,2,\dots,l_B,l_B+1\}$.
- For each letter $a\in\Sigma$, such that $(F_a)_{i,j}\neq\varepsilon$ a transition from i to j exists with the label a. The transition will be noted $(i\xrightarrow{a}j)$.
- One associates with each transition $(i\xrightarrow{a}j)$ a weight defined as $(F_a)_{i,j}$.

(1) Let us prove first that the set of coefficients $(1,j)$ of any matrix product $\{(F_{a_1}F_{a_2}\cdots F_{a_k})_{1,j},\ a_1a_2\cdots a_k\in\Sigma^*\}$ is finite.
- $(F_{a_1}F_{a_2}\cdots F_{a_k})_{1,j}$ is the maximum weight for any path with label $a_1a_2\cdots a_k$ from 1 to j.
- Let V_j be the set of possible values for the weight of any elementary path from 1 to j. Since the number of elementary paths is finite, the set V_j is finite.
  Let M_j and m_j be the maximal and minimal values of V_j.
- Let V_j' be the set of possible values for the weight of any path from 1 to j. Loops can now appear in such paths. One path with loops can be decomposed into a set of loops and a path without loops.
  In the graph, all loops (except the loop from 1 to 1) have the same weight: δ^{-1}. Since the first loop has a null weight, one can consider the path which does not include this first loop.
  The weight of any path with loops is equal to the weight of a path without loops plus a multiple of δ^{-1}.

  $$V_j'\subset(V_j\cup\delta^{-1}\otimes V_j\cup\delta^{-2}\otimes V_j\cup\cdots),$$

  where $\delta^{-k}\otimes V_j$ is the set of values of V_j multiplied by δ^{-k}.
  An upper bound of V_j' is M_j the maximum of V_j.
- Consider a path from 1 to j with k weighted loops. Let c be the weight of this path. Let $k_0^j=\lfloor(M_j-m_j)/\delta\rfloor+1$. As soon as the path goes through more than k_0^j weighted loops, the weight of this path is less than m_j. Indeed $c=c_e-k\times\delta$ with $c_e\in V_j$. $c<M_j-(M_j-m_j)=m_j$.
  For the same label, another path with better weight exists: all loops are in the beginning of the path, with no more weighted loops. The weight of this path belongs to V_j, so it is greater than m_j.
- One then has to consider only paths which have no more than k_0^j weighted loops.
  There are only a finite number of paths with at most k_0^j loops with weight δ^{-1}. The null weighted loops do not influence the path weight.

So the set of weights of optimal paths from 1 to j for any word $a_1a_2\cdots a_k$ belongs to

$$V'_j \subset V''_j = V_j \cup \delta^{-1} \otimes V_j \cup \delta^{-2} \otimes V_j \cup \cdots \cup \delta^{-k_0^j} \otimes V_j,$$

V''_j is finite, then V'_j is also finite.

(2) Let us prove now that $\{(X_0F_{a_1}F_{a_2}\cdots F_{a_k})_l,\ a_1a_2\cdots a_k \in \Sigma^*\}$ is finite. Let w be the word $a_1a_2\cdots a_k$.

- The purpose is to compute $X_0\psi(w)$ where the application ψ is the unique morphism such that $\forall a \in \Sigma,\ \psi(a) = F_a$.

$$(X_0\psi(w))_l = \bigoplus_{i=1}^{l_B+1} X_0(i)(\psi(w))_{i,l}$$

corresponds to the maximal weight of paths with label w, from i to l, where a path going out of i has a penalty weight $X_0(i)$.

- The set $U = \{(X_0\psi(w))_l,\ w \in \Sigma^*\}$ has an upper bound.
An upper bound of set U is $\max_{i=1}^{l_B+1}(X_0(i))+K$, where K is the maximal weight of elementary paths.
- The set U has a lower bound.

$$(X_0\psi(w))_l \geqslant X_0(1)(\psi(w))_{1,l} \geqslant X_0(1)m_l.$$

- $U = \{(X_0\psi(w))_l,\ w \in \Sigma^*\}$ is finite.
$(X_0\psi(w))_l$ can be decomposed into the weights of an elementary path, some loops and a penalty.

$$\begin{aligned}(X_0\psi(w))_l &= X_0(i)\psi(w)_{i,l}\\ &= X_0(i)\delta^{-k}c_e,\end{aligned}$$

where c_e is the weight of the elementary path. Since U is bounded, k is bounded. c_e and $X_0(i)$ belong to finite sets, then U is finite.

Since $\{(X_0\psi(w))_l,\ w \in \Sigma^*\}$ is finite, $\{X_0\psi(w),\ w \in \Sigma^*\}$ is also finite. □

Corollary 7. *Let* $X_0 = (e, \delta^{-1}, \delta^{-2}, \ldots, \delta^{-l_B})$. *The set* $\{X_0F_{a_1}F_{a_2}\cdots F_{a_k},\ a_1a_2\cdots a_k \in \Sigma^*\}$ *is finite.*

Corollary 8. *The orbit automaton for the best occurrence problem is finite.*

The size of set $\{X_0F_{a_1}F_{a_2}\cdots F_{a_k},\ a_1a_2\cdots a_k \in \Sigma^*\}$ is clearly greater in the case of the best occurrence of a word than in the NW case (compare Tables 1 and 2). This comes from the fact that in the best occurrence case the relation $\sim$ is no longer useful. No element of the set $\{X_0F_{a_1}F_{a_2}\cdots F_{a_k},\ a_1a_2\cdots a_k \in \Sigma^*\}$ can be $(max, +)$ proportional to another vector in this same set (the first coordinate of each vector in this set is always e, and if two vectors are proportional, they are equal).

Table 2
Sizes of orbit automata generated by $(\max, +)$ matrices for the best occurrence algorithm. The modified matrices are the same as the original ones except the coefficient $(l_B + 1, l_B + 1)$ which is equal to δ^{-1} instead of e

Words	Original Matrices		Modified Matrices	
	Orbit automaton size	Depth	Orbit automaton size	Depth
AAAA	164	14	87	11
ATTA	305	12	139	9
ATCG	365	10	191	8
ATCGA	1680	13	599	12
ATCGAT	6162	16	1840	13
ATCGATC	23116	18	5489	16
ATCGATCG	79205	23	15842	19

Corollary 9. *Let $B \in \Sigma^*$. The score of the best occurrence of B in any sequence can take only a finite number of values.*

Proof. In the orbit automaton all transitions have a null weight. The proportionality coefficient $(\bigotimes_{k=1}^{l_A} \lambda_k)$ is null. There are no more values than states in this automaton. □

The automaton size can be reduced. When building matrices C'_n, two e have been introduced in position $(1,1)$ and $(l_B + 1, l_B + 1)$, corresponding respectively to non-penalty of insertions/deletions at the beginning and end positions in target sequences. The last element can be kept to δ^{-1}, as in the NW case. Then the penalties at the end positions are no longer null.

If one builds the automaton associated with these new biased matrices, the automaton has fewer states (compare columns of Table 2). But an additional variable is necessary when going through the automaton to obtain the right score. At each stage the algorithm keeps the maximum of scores obtained up to this stage. When the whole sequence has been read, this maximum corresponds to the score of the best occurrence of the query sequence. This computation is $O(l_A)$, i.e. of the same complexity as the scanning process.

7. Smith and Waterman algorithm

The same analysis can be done for the Smith and Waterman algorithm, but we have to remember during this new section that the SW score is not necessary in the last row or column of the classical dynamic programming table. Let us consider the linear penalty function for insertions/deletions. The basic recurrence (Eq. (3)) becomes

$$S(i,j) = (S(i-1,j-1) \otimes \sigma_{i,j}) \oplus (\delta^{-1} \otimes S(i-1,j)) \oplus (\delta^{-1} \otimes S(i,j-1)) \oplus e.$$

Let Y_n be the row vector of size l_B+1 corresponding to the nth row of the classical dynamic programming table. We have

$$Y_n = Y_{n-1}C''_n \oplus Y_n D_n \oplus T$$

with

$$C''_n = \left(\begin{array}{c|ccccc} e & \sigma_{1,n} & . & \cdots & . \\ \hline . & \delta^{-1} & \sigma_{2,n} & . & . \\ . & . & \delta^{-1} & \ddots & . \\ . & . & . & \ddots & \sigma_{l_B,n} \\ . & . & . & . & \delta^{-1} \end{array}\right), \quad D_n = \begin{pmatrix} . & \delta^{-1} & . & . & . \\ . & . & \delta^{-1} & . & . \\ . & . & . & \ddots & . \\ . & . & . & . & \delta^{-1} \\ . & . & . & . & . \end{pmatrix}, \quad T^t = \begin{pmatrix} e \\ e \\ \vdots \\ \vdots \\ e \end{pmatrix}.$$

The coefficient $(1,1)$ of matrix C''_n is equal to e to impose $Y_n(0)=0,\ \forall n\in[1,l_A]$. One can change the definition of matrices C''_n by setting $C''_n(1,1)=\delta^{-1}$. This does not influence the property $Y_n(0)=0,\ \forall n\in[1,l_A]$. In such a case the 0 of the first coordinate will be imposed by comparison to vector T, of which the first term is 0.

D_n is independent of n, and D^* is defined in the same way (Eq. (8)). We have $TD^*=T$. Let $G_n=C''_nD^*$. The matrix G_n is almost the same as F_n (Eq. (10)). The only difference: the coefficients $(1,1)$ and (l_B+1,l_B+1) are now equal to δ^{-1}. The next recurrence stands:

$$Y_0=(e,e,\ldots,e),$$
$$Y_n=Y_{n-1}G_n\oplus TD^*=Y_{n-1}G_n\oplus T.$$

This is an *affine dynamic system*. The first iterations are:

$$Y_1=Y_0G_1\oplus T,$$
$$Y_2=Y_1G_2\oplus T=Y_0G_1G_2\oplus TG_2,\oplus T,$$
$$Y_n=\bigoplus_{k=1}^{n+1} T\bigotimes_{i=k}^{n} G_i$$

with the convention: $\bigotimes_{i=n+1}^{n} G_i=I$ where I is the *identity matrix* for $\mathbb{R}_{\max}^{(l_B+1)\times(l_B+1)}$.

The vectors $T\bigotimes_{i=k}^{n} G_i$ are associated with suffixes of substring $A[1,n]$. The previous computation corresponds to searching for the suffix with maximum score among all suffixes of the word $A[1,n]$. This computation has to be performed for each prefix $A[1,k],\ \forall k\in[1,l_A]$. In the case of the SW algorithm one has to retain the maximal score among all values in the table. This can be written by

$$SW=\bigoplus_{k=0}^{l_A}|Y_k|,\quad \text{where } |Y_k|=\bigoplus_{i=1}^{l_B+1}Y_k(i). \tag{11}$$

The score is the maximum of norms of vectors Y_k, where norm is defined by the maximum of the coordinates of the vector (Eq. (11)).

Linear rewriting: If one increases the size of the system, the system becomes linear. Indeed, each term of vectors $(Y_l)_{l=1\ldots l_A}$ is compared to e. Adding one coordinate which is null, we have

$$(e, Y_n) = (e, Y_{n-1}) \left(\begin{array}{c|c} e & (e,\ldots,e) \\ \hline \varepsilon & \\ \vdots & G_n \\ \varepsilon & \end{array}\right).$$

Let $\{\tilde{G}_a,\ a \in \Sigma\}$ be the set of matrices of size l_B+2 obtained by adding this coordinate.

Theorem 10. *Let X_0 be the unit vector of dimension l_B+2: $X_0=(e,e,\ldots,e)$. The set $\{X_0\tilde{G}_{a_1}\tilde{G}_{a_2}\cdots\tilde{G}_{a_k},\ a_1a_2\cdots a_k \in \Sigma^*\}$ is finite.*

Proof. All generator matrices are triangular. The diagonal terms are all negative except the first one which is null. So the same method as for Theorem 6 gives the result. □

In this formulation one computes all values of the classical dynamic programming table, but one does not retain the maximal score among all terms. Then one introduces an additional variable to *memorize* at each stage, the maximal value obtained up to this stage. Let $Y'_n=(e, Y_n, M_{n-1})$ where M_{n-1} is this *memory* which represents the best score of local alignment between sequence B and $A[1, n-1]$: $M_n = \max(e, M_{n-1}, |Y_n|)$. The next relation stands:

$$(e, Y_n, M_{n-1}) = (e, Y_{n-1}, M_{n-2}) \left(\begin{array}{c|c|c} e & (e,\ldots,e) & e \\ \hline \varepsilon & & e \\ \vdots & G_n & \vdots \\ \varepsilon & & e \\ \hline \varepsilon & (\varepsilon,\ldots,\varepsilon) & e \end{array}\right).$$

In the previous equation there is a gap between the index of the vector Y_n and the index of memory M_{n-1}. Let $\tilde{Y}_n=(e, Y_n, M_n)$. We have

$$\tilde{Y}_n = \tilde{Y}_{n-1} \underbrace{\left(\begin{array}{c|c|c} e & (e,\ldots,e) & \varepsilon \\ \hline \varepsilon & & \varepsilon \\ \vdots & G_n & \vdots \\ \varepsilon & & \varepsilon \\ \hline \varepsilon & (\varepsilon,\ldots,\varepsilon) & e \end{array}\right)}_{H_n} \oplus \tilde{Y}_n \underbrace{\left(\begin{array}{c|c|c} e & (\varepsilon,\ldots,\varepsilon) & e \\ \hline \varepsilon & & e \\ \vdots & I & \vdots \\ \varepsilon & & e \\ \hline \varepsilon & (\varepsilon,\ldots,\varepsilon) & e \end{array}\right)}_{K}, \tag{12}$$

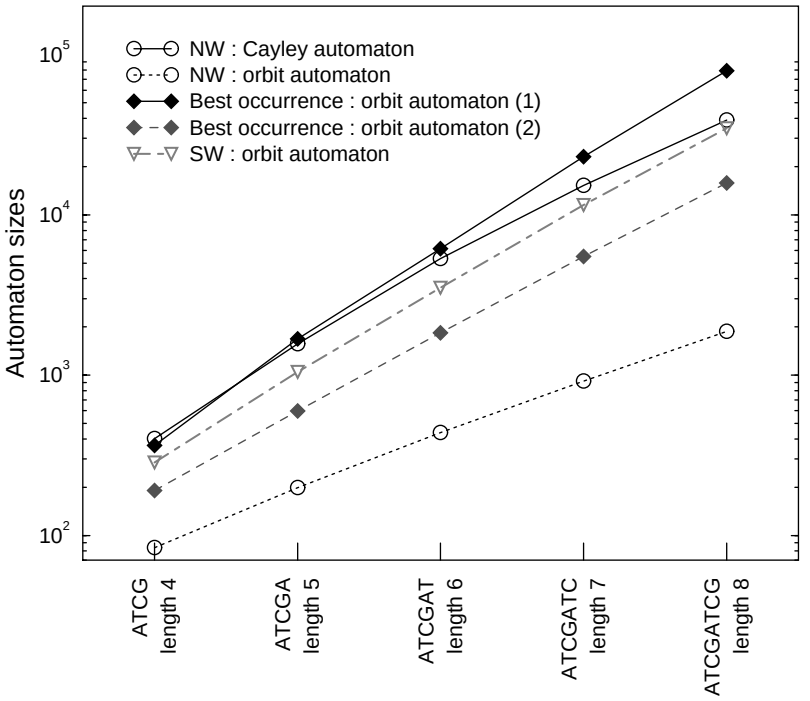

Fig. 2. Comparison of automaton sizes associated with the algorithms: Needleman and Wunsch (NW), best occurrence and Smith and Waterman (SW). For NW, Cayley automata and orbit automata have been considered. For the best occurrence algorithm both curves represent sizes of orbit automata, the first is associated with the original matrices (the coefficient (l_B+1, l_B+1) is equal to 0), the second with the modified matrices (the coefficient (l_B+1, l_B+1) is equal to δ^{-1}).

K is independent of n. The graph associated with K has no circuit with positive weight. The matrix K^* exists. K is idempotent, so we have $K = K^2$ and $K^* = K$. The solution of Eq. (12) is given by

$$\tilde{Y}_n = \tilde{Y}_{n-1} \underbrace{H_n K^*}_{L_n}.$$

The expression of the SW problem is exactly the same as for the NW problem and the formulation with automata is the same. Only the matrices differ and the dimension of the problem has been increased. The automaton is then expected to be much bigger than for the best occurrence algorithm. It is the case when the matrices for the best occurrence algorithm have been modified but the automaton is smaller if original matrices have been chosen (Compare Tables 2 and 3). The reason is that one does not have to compare and store the comparison between two subsequences without any similarity, contrary to the case of the best occurrence. Fig. 2 represents the automaton sizes for each type of algorithm.

Table 3
Sizes of orbit automata generated by $(\max,+)$ matrices for SW algorithm

Words	Orbit automaton size	Depth
AAAA	86	11
ATTA	190	14
ATCG	286	10
ATCGA	1048	12
ATCGAT	3513	15
ATCGATC	11546	17
ATCGATCG	34624	23

8. Affine penalty functions

The general case of dynamic programming sequence alignment allows affine penalty functions for gaps. The penalty function for a gap of length k is defined by: $g(k) = go + ge \times (k-1)\ \forall k \geqslant 1$. Moreover we suppose $go \geqslant ge$.

Let us consider the NW case. The SW case can be easily deduced. The initial recurrence is given by Eq. (2). Let X_n and Y_n be the row vectors defined by

$$X_n(l) = N(n,l),$$

$$Y_n(l) = \max_{1 \leqslant k < n} (X_{n-k}(l) - (go + (k-1)ge)).$$

The vector X_n represents the row n of the classical dynamic programming table. $X_n(l)$ is the maximal value for aligning sequences $A[1,n]$ and $B[1,l]$. The values $Y_n(l)$ are the maximal values for aligning $A[1,n]$ and $B[1,l]$ with an insertion in sequence A at the end of the alignment.

$$\begin{aligned} Y_n(l) &= \max_{1 \leqslant k < n} (X_{n-k}(l) - (go + (k-1)ge)) \\ &= \max\left(\max_{2 \leqslant k < n} (X_{n-k}(l) - go - (k-1)ge), X_{n-1}(l) - go \right) \\ &= \max(Y_{n-1}(l) - ge, X_{n-1}(l) - go), \end{aligned}$$

$$X_n(l) = \max \begin{pmatrix} X_{n-1}(l-1) + \sigma_{l,n}, \\ Y_n(l), \\ \max_{1 \leqslant k \leqslant l} (X_n(l-k) - go - (k-1)ge) \end{pmatrix}.$$

We can rewrite these equations in term of vectors X_n and Y_n:

$$X_n = X_{n-1}C_n \oplus X_n D \oplus Y_n,$$

$$Y_n = Y_{n-1}E \oplus X_{n-1}F$$

with

$$C_n = \begin{pmatrix} . & \sigma_{1,n} & . & . & . \\ . & . & \sigma_{2,n} & . & . \\ . & . & . & \ddots & . \\ . & . & . & . & \sigma_{l_B,n} \\ . & . & . & . & . \end{pmatrix},$$

$$D = \begin{pmatrix} . & -go & -go-ge & -go-2ge & \cdots & -go-(l_B-1)ge \\ . & . & -go & -go-ge & \cdots & -go-(l_B-2)ge \\ . & . & . & -go & \cdots & -go-(l_B-3)ge \\ . & . & . & . & \ddots & \vdots \\ . & . & . & . & . & -go \\ . & . & . & . & . & . \end{pmatrix},$$

$$E = -ge \otimes I,$$

$$F = -go \otimes I,$$

where I is the *identity matrix* for $\mathbb{R}_{\max}^{(l_B+1)\times(l_B+1)}$. The graph associated with D has no circuit with positive weight, so D^* exists and can be easily computed. The previous equations can be summarized by the following:

$$(X_n, Y_n) = (X_{n-1}, Y_{n-1}) \left(\begin{array}{c|c} (C_n + F)D^* & F \\ \hline ED^* & E \end{array} \right).$$

Exactly the same recurrence is valid for the problem of the best occurrence of a word, only initialization changes. In the case of the Smith and Waterman algorithm we need a memory as in the linear case, but the final recurrence is the same in spirit.

9. Applications

The large size of the automaton is the limitation of this new algorithm. Even in simple cases considered below (with affine penalty functions and simple substitution matrices) the automaton size is quite large. Fig. 2 indicates that the automaton size grows exponentially in the length of the query sequence. Each additional letter multiplies the automaton size by a factor between 2 and 4, depending on the algorithm: NW score, best occurrence or SW score. This fact rules out the hope of comparing proteins (whose lengths range from one hundred to several thousands) with complex substitution matrices. However, biologists frequently face the problem of comparing

one short sequence against many other strings. This problem is a generalization of pattern matching in which each mutation has a specific cost depending on the letters involved.

The classical problem of pattern matching with error can be solved by a variation of the dynamic programming Needleman and Wunsch algorithm: the gap penalties are $go = ge = 1$, the substitution matrix is built with 0 on the diagonal and with -1 elsewhere. The process retains the maximum of scores when comparing the query sequence and each subsequence of the target sequence. The initializations are strictly the same as in Eq. (9) and the transfer matrices are given by Eq. (10). The score is obviously negative or null; it is null iff the word appears exactly in the sequence. The absolute value of the score is also the number of errors between the word and the best occurrence of this word. In that case, the first coordinate of vectors is always 0 (cf. Section 6). So, our algorithm with an automaton does not have to consider the $(max, +)$ proportionality of vectors during the construction of the automaton. Equality is the only interesting case. This constraint is easier to implement.

The algorithm decomposes into two stages: building the automaton and databank scanning. The second stage is very fast since the scanning process has linear time complexity in the size of the databank. The algorithm to build the automaton is more complex. The size of the finite set we are interested in is not known before building the automaton. However, one can give the complexity of this stage dependent on the size of automaton N_{au}. For each state in the automaton one has to consider all possible transitions from this state, then compare all new states with all states already built. Therefore, the time complexity is lower than $O(N_{\mathrm{au}}^2.|\Sigma|/2)$. With a hash function all these comparisons can be done faster.

Orbit automaton algorithm and dynamic programming algorithm for pattern matching with errors have been implemented in LASSAP [11], an intra- and inter-databases high performance search engine. Table 4 summarizes the gains of our new implementation of dynamic programming in the case of pattern matching with errors. The results for the automaton algorithm are satisfactory for short words. If the automaton is stored, column (2) shows the time of *databank scanning* which means the time necessary to scan the entire databank. Theoretically, databank scanning time would be constant. Observed variations come from the memory management. The bigger the automaton, the more numerous are the memory rearrangements.

The time necessary to complete the first stage (automaton building) is not constant. The explosion of time for building the automaton is due to the explosion of automaton size. Indeed, with the hash function of the present implementation, one observes for the algorithm of automaton construction, a time complexity almost linear in automaton size.

It seems that the automaton size is an exponential function of sequence length, but it depends on the number of generators in the semigroup of matrices. Generally the number of generators is equal to the alphabet size. In our application the substitution matrix is very simple: a positive constant on the diagonal and a negative one elsewhere. In that case, if all letters of the alphabet do not appear in the query sequence, there is one more generator than there are different letters; otherwise the number of generators is the

Table 4
Performances of the orbit automaton algorithm compared with the classical dynamic programming algorithm. For each word and for each target sequence in the SwissProt databank Rel. 34, one seeks the subsequence of the target sequence that minimizes the number of errors when aligned. Both algorithms are implemented in LASSAP. Results have been obtained on a SUN 4000, using only one processor. Speeds represent the ratio MMC/s (million matrix cells per second)

Words	Length	Dynamic programming		Automaton			
		Time (s)	Speed	Time (s)			Automaton size
				Total	(1)	(2)	
BAAABF	6	15.20	8.37	9.795	0.005	9.79	83
KIIKLHEN	8	19.93	8.51	9.86	0.034	9.826	472
VKIIKLHEN	9	22.25	8.58	10.284	0.087	10.197	1114
AASDTGSTYL	10	24.48	8.66	10.442	0.18	10.262	2397
LVIVSVFDLAS	11	26.68	8.74	10.885	0.405	10.48	4928
KNVIGARRASWR	12	28.69	8.87	12.176	1.237	10.939	12033
RAANQDYVITRTN	13	31.11	8.86	13.935	2.716	11.219	24331
QGQQFPNECQLDQL	14	33.16	8.95	17.389	6.089	11.30	50820
QGQQFPNECQLDQLN	15	35.49	8.96	25.916	13.098	12.818	107408

(1): Construction of automaton.
(2): Scanning process.

size of the alphabet. For example, the protein sequence BAAABF is composed of three different letters. The number of generators is 4: three matrices for the distinct letters appearing in the query sequence plus one matrix for any letter not present in the query sequence.

The number of generators is crucial for the automaton size. For queries of the same length, a greater number of generators leads to a bigger automaton (cf. Fig. 3).

In the current implementation there is an optimal word length for using this algorithm. According to the difference between the execution times with the classical algorithm and the automaton algorithm, the gain is maximal for a sequence of length 12 or 13. This algorithm is then interesting if the query sequence is short ($l_B \leqslant 13$) and if the databank is very large. The time necessary to build automata must be distributed over an enormous set of target sequences. When databanks increase, the length of the query sequence can increase while gaining in computing time.

The same kind of ideas has been explored by Esko Ukkonen who has implemented the *pattern matching* with errors with automata [24]. He focused only on the occurrences with at most k errors. His algorithm consists of building a finite deterministic automaton. Each state corresponds intuitively to a row of the classical dynamic programming table. He proposed to reduce the number of states taking into account that the occurrences with more than k errors are not interesting at all. In spirit, his algorithm is close to ours but our method can be applied with any substitution matrix and is the same for pattern matching, NW score and SW score.

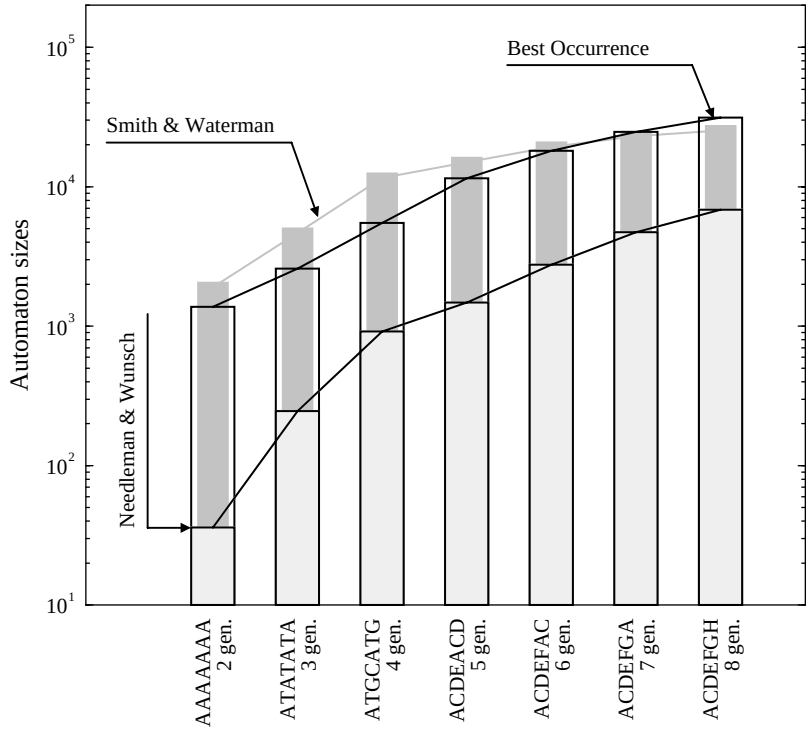

Fig. 3. Influence of the number of generators on the size of orbit automaton associated with the algorithms: Needleman and Wunsch, best occurrence and Smith and Waterman. All query sequences have a length equal to 7. The number of generators increases from 2 to 8. For the best occurrence algorithm we considered only the modified matrices.

10. Discussion

This algorithm may not appear to be practical because it seems difficult to adapt it for comparing proteins whose lengths range up to several thousands. Actually a version of this algorithm is implementable for real sequences. The present implementation splits the stage of automaton building from the stage of scanning. Another possibility is to build automaton states when needed. Constructing the automaton on the fly would accelerate the global process since some states can never be used. Remember that during the process of automaton construction one has to compute the vectors corresponding to all possible lines of the classical dynamic programming table. If some associated states are never used, these computations are useless and a pure waste of time. The on the fly construction of automaton allows one to do only necessary computations, computing only necessary states. In that sense the algorithm is optimized.

This algorithm would work as follows. Let us consider that the automaton is not yet totally built and that the scanning process skims a particular target sequence. If the current configuration has appeared before (i.e. if the scanning process goes through a

previously computed transition), the algorithm takes advantage of the previous computations and does not carry out the same ones again. On the other hand, if the current letter does not correspond to a previously computed transition, a new state is computed. That computation is equivalent in terms of complexity to the computation of the row in the classical dynamic programming algorithm. Then the process has to compare this new state to all previously built ones. This task can be done efficiently and almost independently of the automaton size with a hash function.

The space complexity due to the automaton size then becomes the main challenge. This limitation can be overcome by fixing a maximum size for the automaton. If this limit is not reached, the new state is computed, otherwise the algorithm goes back to the classical dynamic programming algorithm to carry out the remainder of the computations. Then the challenging question is the strategy for choicing the states which have to be integrated into the automaton. This on the fly algorithm is implementable and would not have a total time complexity greater than the classical one.

Acknowledgements

The author is much indebted to S. Gaubert for suggesting this application and for useful discussion, to J.-J. Codani and E. Glémet for the use of LASSAP [11] to implement this automaton algorithm in large scale environment.

References

[1] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, D.J. Lipman, Basic local alignment search tool, J. Mol. Biol. 215 (1990) 403–410.
[2] S.F. Altschul, T.L. Madden, A.A. Schäffer, J. Zhang, Z. Zhang, W. Miller, D.J. Lipman, Gapped BLAST and PSI-BLAST: a new generation of protein database search programs, Nucleic Acids Res. 25 (17) (1997) 3389–3402.
[3] F. Baccelli, G. Cohen, G.J. Olsder, J.-P. Quadrat, Synchronization and Linearity, Wiley, New York, 1992.
[4] R. Baeza-Yates, G.H. Gonnet, A new approach to text searching, Commun. ACM 35 (10) (1992) 74–82.
[5] R. Baeza-Yates, C.H. Perleberg, Fast and practical approximate string matching, in: CPM'92, Lecture Notes in Computer Science, vol. 664, Springer, Berlin, 1992, pp. 185–192.
[6] R.S. Boyer, J.S. Moore, A fast string searching algorithm, Commun. ACM 20 (10) (1977) 762–772.
[7] C. Choffrut, Contributions à l'étude de quelques familles remarquables de fonctions rationnelles, Thèse de doctorat d'Etat, Université Paris 7, LITP, Paris, France, 1978.
[8] V. Froidure, J.-E. Pin, Algorithms for computing finite semigroups, in: F. Cucker, M. Shub (Eds.), Foundations of Computational Mathematics, Lecture Notes in Computer Science, Springer, Berlin, 1997, pp. 112–126.
[9] S. Gaubert, Théorie des systèmes linéaires dans les dioïdes, Ph.D. Thesis, Ecole des mines de Paris, France, 1992.
[10] S. Gaubert, Performance evaluation of $(\max, +)$ automata, IEEE Trans. Automat. Control 40 (12) (1995).
[11] E. Glémet, J.-J. Codani, LASSAP: a large scale sequence comparison package, Comp. Appl. BioSci. 13 (2) (1997) 137–143.
[12] M. Gondran, M. Minoux, Graphs and Algorithms, Wiley, New York, 1984.

[13] O. Gotoh, An improved algorithm for matching biological sequences, J. Mol. Evol. 162 (1982) 705–708.
[14] D.E. Knuth, J.H. Morris, V. Pratt, Fast pattern matching in string, SIAM J. Comput. 6 (1977) 323–350.
[15] D.J. Lipman, W.R. Pearson, Rapid and sensitive protein similarity searches, Science 227 (1985) 1435–1441.
[16] W. Miller, E.W. Myers, Optimal alignments in linear space, CABIOS 4 (1) (1988) 11–17.
[17] W. Miller, E.W. Myers, Sequence comparison with concave weighting functions, Bull. Math. Biol. 50 (2) (1988) 97–120.
[18] M. Mohri, Finite-state transducers in language and speech processing, Comput. Linguistics 23 (2) (1997).
[19] S.B. Needleman, C.D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, J. Mol. Biol. 48 (1970) 443–453.
[20] P.H. Sellers, On the theory and computation of evolutionary distances, SIAM J. Appl. Math. 26 (1974) 787–793.
[21] T.F. Smith, M.S. Waterman, Comparison of bio-sequences, Adv. Appl. Math. 2 (1981) 482–489.
[22] T.F. Smith, M.S. Waterman, Identification of common molecular subsequences, J. Mol. Biol. 147 (1981) 195–197.
[23] J. Tarhio, E. Ukkonen, Boyer–Moore approach to approximate string matching, in: J.R. Gilbert, R.G. Karlsson (Eds.), Proc. SWAT'90, Lecture Notes in Computer Science, vol. 447, Springer, Berlin, Bergen, Norway, July 1990, pp. 348–359.
[24] E. Ukkonen, Finding approximate patterns in strings, J. Algorithms 6 (1985) 132–137.
[25] M.S. Waterman, M. Eggert, A new algorithm for best subsequence alignments with application to tRNA-tRNA comparisons, J. Mol. Biol. 197 (1987) 723–728.
[26] S. Wu, U. Manber, Agrep—A fast approximate pattern-matching tool, in: Usenix Winter 1992 Technical Conf., San Francisco, January 1992, pp. 153–162.
[27] S. Wu, U. Manber, Fast text searching allowing errors, Commun. ACM 35 (10) (1992) 83–91.
[28] S. Wu, U. Manber, A fast algorithm for multi-pattern searching, Technical Report TR 94-17, University of Arizona at Tuscon, May 1994.