CHAPTER 5

Inference of Network Expressions

5.0. Introduction

This chapter introduces various mathematical models and combinatorial algorithms that are used to infer network expressions which appear repeated in a word or are common to a set of words, where by *network expression* is meant a regular expression without Kleene closure on the alphabet of the input word(s). A network expression on such an alphabet is therefore any expression built up of concatenation and union operators. For example, the expression $\mathtt{A}(\mathtt{C}+\mathtt{G})\mathtt{T}$ concatenates A with the union $(\mathtt{C}+\mathtt{G})$ and with T. Inferring network expressions means *discovering* such expressions which are initially unknown. The only input is the word(s) where the repeated (or common) expressions will be sought. This is in contrast with another problem, we shall not be concerned with, which *searches* for a known expression in a word(s) both of which are in this case part of the input. The inference of network expressions has many applications, notably in molecular biology, system security, text mining, etc. Because of the richness of the mathematical and algorithmic Al problems posed by molecular biology, we concentrate on applications in this area. The network expressions considered may therefore contain spacers where by *spacer* is meant any number of do not care symbols (a do not care is a symbol that matches anything). *Constrained spacers* are consecutive do not care symbols whose number ranges over a fixed interval of values. Network expressions with do not care symbols but no spacers are called "simple" while network expressions with spacers are called "flexible" if the spacers are unconstrained, and "structured" otherwise. Both notions are important in molecular biology.

Applied Combinatorics on Words, ed. David Tranah.
Published by Cambridge University Press.

Applications to biology motivate us also to consider network expressions that appear repeated not exactly but approximately.

Only exact combinatorial methods that are nontrivial (that is, are not simple brute-force schemes which enumerate all possible network expressions) will be mentioned. In most cases, the network expressions that have been considered in the literature present some constraint that generally applies to the union operator. Indeed, the operands concerned by the union operation will most often be elements of the alphabet $\mathcal{A}$ and not arbitrary words in $\mathcal{A}^+$ as is the case with unrestricted network expressions. For instance, we do not allow expressions such as $\mathtt{A(CG+G)T}$.

The literature on the inference of regular expressions, also called grammatical inference, is vast, and predates computational biology by many years. The inference problems addressed in this chapter present special characteristics in relation to such general problems. The most important ones are that, although the expressions considered here are simpler in the sense indicated above, their occurrences are not exact and come hidden inside often very large texts. To use the terms commonly adopted in the grammatical inference community, we work with (positive) examples that have first to be fished from a sea of other textual information, mostly noise. Most often, there is not only one regular expression, and thus one set of examples, but various distinct ones hidden in the same text.

5.1. Inferring simple network expressions: models and related problems

5.1.1. The star model

A *star expression* X is an expression of the form $X = e_1 e_2 \cdots e_m$ where each e_i is the union of elements of the alphabet, that is $e_i = a_{i,1} + a_{i,2} + \cdots + a_{i,n_i}$ with $a_{i,j} \in \mathcal{A}$ for $1 \leq j \leq n_i \leq \mathrm{Card}(\mathcal{A})$, $1 \leq i \leq m$. Star expressions are therefore words on the alphabet $\mathcal{P}(\mathcal{A})$ of all nonempty subsets of $\mathcal{A}$, that is, they are elements of $\mathcal{P}(\mathcal{A})^+$. This includes the set $\mathcal{A}$ which we denote by $\bullet$ and call the *do not care* symbol.[1] Let $F(w)$ denote the set of factors of a word w. A star expression denotes also a finite set of words of length m. A star expression $X \in \mathcal{P}(\mathcal{A})^+$ is said to occur *exactly* in a word $w \in \mathcal{A}^+$ if there exists $v \in F(w)$ such that $v \in X$. The factor v is said to be an *occurrence* of X in w.

The notion of approximate occurrence relies on the notion of a distance between two words u and v on $\mathcal{A}$. In biology, like in a number of other text applications, a natural distance measures the effort required to transform one word into the other given certain allowed operations. The operations

[1] One also finds in the literature the terms *wild card* and *joker*.

that model best the mutational events that may happen during replication and survive are the *substitution* (that is replacement) of a letter of $\mathcal{A}$ by another, the *deletion* of a letter in one of the two words, and the *insertion* of a letter. Finally, a *match* is an operation that leaves the letter unchanged. These are called *edit operations*.

Let S, D, I, M denote the four edit operations described above. An *edit transcript* is a string over the alphabet S, D, I, M that describes a transformation of a word u into another v. An equivalent way of describing such transformation is through a *global alignment* of u and v. This is obtained by inserting spaces in both u and v, transforming them into u' and v' defined over $\mathcal{A} \cup \{-\}$ where $\{-\}$ denotes a space and $|u'| = |v'|$. An edit transcript can be easily converted into a global alignment and viceversa.

A cost c may be attributed to each operation where c is a function $(\mathcal{A} \cup \{-\}) \times (\mathcal{A} \cup \{-\}) \to \mathbb{R}$. The cost of a global alignment (and thus of the associated edit transcript) of two words u' and v' of length n is then $\sum_{i=0}^{n-1} c(u'[i], v'[i])$. Not all cost functions define a distance. This will be the case if the function c is symmetric and if $c(a, b)$ for $a, b \in \mathcal{A} \cup \{-\}$ is strictly greater than 0 if $a \neq b$ and is 0 otherwise.

Two types of distances have attracted special attention. These are the *Hamming* distance and the *edit* distance (this last is also called the *Levenshtein* distance).

In the case of the edit distance, the cost function is

$$c(a, b) = \begin{cases} 1 & \text{if } a \neq b; \\ 0 & \text{otherwise} \end{cases} \quad \text{for } a, b \in \mathcal{A} \cup \{-\}.$$

The edit distance is thus the minimum number of substitutions, insertions, and deletions required to transform u into v. The Hamming distance applies only to words of the same length, and is restricted to substitution/match operations. The cost function is the same as for the edit distance applied to $a, b \in \mathcal{A}$. It counts the minimum number of substitutions needed to obtain v from u assuming they have the same length. The Hamming distance will be denoted by $dist_H$ and the edit distance by $dist_E$.

Given a positive integer d, a d-occurrence of X in a word w is a word $v \in F(w)$ such that there exists a word $u \in X$ with $dist(u, v) \leq d$ for some fixed d. An expression $X \in \mathcal{P}(\mathcal{A})^+$ is thus said to appear *approximately* in w if it has a d-occurrence in w. Where there is no possible ambiguity, reference to d will be dropped in all such notations.

The distances considered between words v and u are, as suggested, usually Hamming or edit, although any other may be used. For ease of exposition, we consider Hamming distance exclusively. Issues related to the use of the edit distance instead are left as open problems in Section 5.5.1.

The above definitions of approximate occurrence of an expression lead to the following inference problem statements. They call upon the concept

of *quorum*. This is the minimum number of times an expression must appear repeated in the input word(s). In the case of a set of words, the quorum is the number of distinct words where the expression appears.

EXPRESSION INFERENCE PROBLEM FOR A STAR EXPRESSION

Expression X repeated in a word

INPUT: A word $w \in \mathcal{A}^+$, a quorum q and a distance d.

OUTPUT: All star expressions $X \in \mathcal{P}(\mathcal{A})^+$ that occur d-approximately in w at least q times.

Expression X common to a set of words

INPUT: N words $w_1, w_2, \ldots, w_N \in \mathcal{A}^+$, a quorum q, a distance d.

OUTPUT: All star expressions $X \in \mathcal{P}(\mathcal{A})^+$ that occur d-approximately in at least q of the N words $w_1, w_2, \ldots, w_N$.

Observe that nothing, except algorithmic Al concerns, would forbid consideration, in all the above definitions and problem statements, of network expressions without constraints on the union operator.

These definitions and statements have been adopted in a number of exact algorithms, namely COMBI, POIVRE, SPELLER, SMILE, PRATT, and MITRA-COUNT. The main difference between the algorithms has been in the type of further constraints put on the network expressions allowed. In COMBI, the expressions are indeed elements of $\mathcal{P}(\mathcal{A})^+$ with just a constraint on the number of times the do not care symbol $\mathcal{A}$ may be used in the expression. This last constraint is used by all algorithms. In POIVRE as in PRATT, the expressions are over an alphabet $\mathcal{S}$ where $\mathcal{S}$ is a proper subset of $\mathcal{P}(\mathcal{A})$. In PRATT furthermore, expressions must appear exactly in a word ($d = 0$). In SPELLER and MITRA-COUNT, the expressions are elements of $\mathcal{A}^+$. SPELLER was later extended to handle elements of $\mathcal{S} \subseteq \mathcal{P}(\mathcal{A})^+$, as is the case for SMILE.

The model described in this section is called a *star* model because the expressions X given as output may be viewed as the centre of star trees whose edges have the distance between X and each of its occurrences in the input word(s) as length. A special case of the problem of finding such expressions has been called the *closest substring problem*. This is stated in the following way, where by $F_k(w)$ we denote the set $F(w) \cap \mathcal{A}^k$ of w having length k.

CLOSEST SUBSTRING PROBLEM

INPUT: N words $w_1, w_2, \ldots, w_N$ over the alphabet $\mathcal{A}$ and integers d and k.

OUTPUT: A word x of length k over $\mathcal{A}$ such that there exist $u_i \in F_k(w_i)$ with $dist_H(x, u_i) \leq d$ for all $1 \leq i \leq N$.

It is interesting to observe the relation between an expression within the star model and another well-known mathematical object: Steiner strings. Given a set of words $x_1, \ldots, x_N$ of the same length in $\mathcal{A}^+$, a *Steiner string* is a word $\bar{x}$ also in $\mathcal{A}^+$ which minimizes $\sum_{i=0}^{N} dist_H(\bar{x}, x_i)$. One may then wonder whether an expression X that solves the *expression inference problem for star expressions* leads to a Steiner string, that is, are the star expressions that are found also solutions of the Steiner string problem for their sets of occurrences? The answer is negative. A simple counterexample is the star expression `ACAA` repeated in the word $w =$ `AAAAAAACAC` with $d = 1$ and $q = 4$. The expression `ACAA` of length 4 is indeed a solution of the star expression inference problem since it has 4 occurrences in w, namely at positions 0, 1, 2, 5 (positions in a word start at 0). Expression `AAAC` is also a solution with 5 occurrences, at positions 0, 1, 2, 3, 5. Neither expression is a Steiner string of its set of occurrences. In both cases, this is the word `AAAA`. Notice that `AAAA` is *not* itself a solution of the problem.

The star-model admits a variant that is interesting for some biological applications. The variant applies to the case of expressions common to a set of words and assumes a phylogenetic tree is given as input together with the words. This is a binary tree that represents the speciation events that have led to the different species currently existing (each represented by a word in the input words set and associated to a leaf of the tree) from the ancestors that are not known. The tree may be unrooted, or rooted if the order of the events in time is known. We assume here that the tree is rooted. It is this tree, and not a star-tree, that is used to compute the distances.

In fact, each edge in the tree is labelled by the Hamming distance between the words at its extremities. Given a phylogenetic tree, a set of words placed at the leaves, and an integer k, factors of length k, one for each input word, and intermediate expressions corresponding to internal nodes have to be inferred such that the sum of the labels over all edges of the tree is minimized. Such minimal sum is called *parsimony score*. The expressions are elements of $\mathcal{A}^+$, the problem (in its decision version) as addressed by the FOOTPRINTER algorithm is as follows.

SUBSTRING PARSIMONY PROBLEM

INPUT: N words $w_1, w_2, \ldots, w_N \in \mathcal{A}^+$, a phylogenetic tree $\mathcal{F}$ for the words, a length k, and an integer d.
OUTPUT: Factors of length k for the leaves and words of length k for all internal nodes of the tree that have a parsimony score of at most d.

Another variant that has been considered constrains the expressions given as solutions to the expression inference problem to satisfy a *uniform*

property. Suppose expressions of a length k are sought and let X be a solution of the problem. Let also two positive integers, $d' < d$ and $k' < k$, be given. The following must then be true: for all i such that $0 \leq i \leq |X|$, $dist_H(X[i\,..\,i+k'-1], v[i\,..\,i+k'-1]) \leq d'$. Intuitively, this constraint imposes that the possible differences between an expression X solution of the problem and each of its occurrences be uniformly spread, hence the name given to the property. A further variant has been used in WEEDER where the constraint that must be satisfied is: for all i such that $0 \leq i \leq |X|$, $dist_H(X[0\,..\,i], v[0\,..\,i]) \leq [(i \times d)/k]$. The inconvenience of a constraint of this latter type is that an asymmetry is introduced: differences may accumulate at the end of the occurrences of an expression.

5.1.2. The clique model

In this model, given an alphabet $\mathcal{A}$, the network expressions on $\mathcal{A}$ that are considered have the constraint that union operators may this time be applied to elements of $\mathcal{A}^k$ only, where k is the length of the expressions sought. Such expressions are therefore collections of words $w \in \mathcal{A}^k$, and the notion of occurrence is associated to both a distance and a quorum. The definition of occurrence in this case is thus operational and includes within it the problem statement.

Such an alternative model has been used by some authors, most notably in the algorithms WINNOWER, MITRA-GRAPH, and KMRC. MITRA-GRAPH uses in fact both models: the clique model and the star. WINNOWER and MITRA-GRAPH formalize the model and problem in graph-theoretical terms.

We state the problem in the case of a set of words. Given a set of N words $w_1, w_2, \ldots, w_N$ over the alphabet $\mathcal{A}$ and two nonnegative integers d and k, let $\mathcal{G} = (V_1 \cup \cdots \cup V_N, E)$ be an N-partite graph where $V_i = F(w_i) \cap \mathcal{A}^k$ (that is it is the set of all factors of length k of w_i for all i) and there is an edge between $v_i \in V_i$, $v_j \in V_j$ for $i \neq j$ if $dist_H(w_i, w_j) \leq 2d$. Given d and k as above, and given a quorum q, we say that a network expression X is a (d, q)-clique if and only if it is a set of q words of length k such that $(u, v) \in E$ for all $u, v \in X$.

The problem is then formulated as follows.

EXPRESSION INFERENCE AS A CLIQUE PROBLEM

INPUT: N words $w_1, w_2, \ldots, w_N \in \mathcal{A}^+$, a quorum q, a length k, and a distance d; the associated N-partite graph $\mathcal{G}$.
OUTPUT: All (d, q)-cliques of $\mathcal{G}$.

The output cliques correspond to expressions X having occurrences that are all pairwise $2d$-approximations[2] of each other in at least q of the N words $w_1, w_2, \ldots, w_N$. Expression X is the union of its occurrences.

In POIVRE instead, the graph is built upon a relation R between the letters of the alphabet $\mathcal{A}$ that is also part of the input. Typically, the relation will be nontransitive and model the degree to which shared physico-chemical properties between the biological units denoted by the letters (nucleic or amino acids) enable them to perform equivalent functions in a molecule. The relation R on the letters of $\mathcal{A}$ is straightforwardly extended to a relation R on words of the same length in $\mathcal{A}^+$ as follows: two factors u, v of length k are in relation by R extended if $u[i]$ is in relation with $v[i]$ by R, for $0 \leq i \leq k-1$. Relation R extended to factors of length k is denoted by R_k. The problem is then expressed in a way that resembles the formulation given above except that graph $\mathcal{G} = (V_1 \cup \cdots \cup V_N, E)$ is now such that there is an edge between nodes $v_i \in V_i$, $v_j \in V_j$ for $i \neq j$ if the corresponding factors in w_i, w_j are in relation by R_k. In POIVRE, the idea was used to infer contiguous motifs in protein structures previously coded into a string of pairs of angles whose values are discretized into integers by means of a grid.

A natural question is whether for each solution X of the expression inference as a clique problem in the case of WINNOWER, there exists an expression Y such that $|Y| = |X|$ and Y is a solution under the star model for the collection of words, which is the set of occurrences of X, distance d, and same quorum. The answer is no. Let us assume expressions in only $\mathcal{A}^+$ are considered. Let the input words be the set $\{w_1 = \texttt{ACAC}, w_2 = \texttt{AGAG}, w_3 = \texttt{ATAT}\}$, $d = 1$ and $q = 3$. The expressions in $\mathcal{A}^+$ of length 4, $X_1 = \texttt{ACAC}$, $X_2 = \texttt{AGAG}$, and $X_3 = \texttt{ATAT}$ are solutions of the *expression inference as a clique problem*, because the (nonproper) factors $w_1 = \texttt{ACAC}$, $w_2 = \texttt{AGAG}$, $w_3 = \texttt{ATAT}$ in the three input words w_i form a clique of the 3-partite graph $\mathcal{G}$. Yet no expression Y in $\mathcal{A}^4$ exists that is a solution of the problem as formulated in the star model for $d/2$. Obviously, there are solutions for distance d. In that case, however, there may be more solutions in the star model, some of which have more occurrences. Consider this time the following set of input words $\{w_1 = \texttt{ACAC}, w_2 = \texttt{AGAG}, w_3 = \texttt{ATAT}, w_4 = \texttt{TCTG}\}$ and d, q as before. Expression $X = \texttt{ACAC|AGAG|ATAT}$ remains a solution within the clique model for quorum 3. Within the star model and with $d/2 = 1$, any expression of the type $\texttt{AbAc}$, with $\texttt{b}, \texttt{c} \in \mathcal{A}$, $\texttt{b} \neq \texttt{c}$, is a solution with three occurrences while expression $Y = \texttt{ACAG}$ is also a solution but with four occurrences (w_1, w_2, w_3, w_4).

[2] It will be explained later in this section why a $2d$ threshold is used instead of simply d.

5.1.3. Other models

Other models have been used, in general, for inferring expressions that are either elements of $\mathcal{A}^+$ or sets of elements of $\mathcal{A}^k$ for a given positive integer k. In the case of expressions in $\mathcal{A}^+$, those sought are the "most surprising" ones in the statistical sense. They correspond to expressions whose probability of occurring (exactly or approximately) in the input word(s) is lower than expected assuming a certain statistical model that is in general a Markov model of order p of the input word(s) for $p < (k-1)$. The algorithms for computing the "most surprising" expressions either perform brute-force enumeration of all possible expressions and then sophisticated exact or approximate statistical evaluation (described in detail in Chapter 6), or are heuristic (for example, PROFILE). We therefore do not speak of this model further.

In the case of expressions that are sets of elements of $\mathcal{A}^k$ for a given positive integer k, it is worth mentioning that another measure has been used to decide whether a set of factors in some input word(s) should be grouped. The measure corresponds to what has been called the *relative entropy* of a set of words or *Kullback–Leibler information number*. This measure is not a distance (triangular inequality is not satisfied) and is global; it is not built upon a pairwise relation between the factors and therefore it does not lead to a graph-theoretical formulation as above. Algorithms that seek network expressions that are elements of $\mathcal{A}^+$ can use the relative entropy of the sets of occurrences of the expressions given as output as a measure of "surprise" that will be different from the measure given by the probability of occurrence of the expression under the same conditions as it was inferred.

5.2. Algorithms

5.2.1. Inference in the star model

Preliminaries: suffix tree and generalized suffix tree

Long words, specially when they are defined over a small alphabet, may contain many exact repetitions. From this observation follows the idea of using an indexed representation of the input word(s). Using a representation that has a unique pointer for identical factors enables comparison of such factors more than once with the expressions as these are inferred to be avoided. The index used by the algorithm whose description follows is a suffix tree.

Details on the suffix tree construction may be found in Chapter 2. We just recall below that the suffix tree of an input word w, denoted by $\mathcal{T}_w$ or simply $\mathcal{T}$ when the input word is clear from the context, has the following properties:

1. each edge of the suffix tree is labelled by a nonempty factor of the input word w;
2. each internal node of the suffix tree has at least two edges leaving it;
3. the factors labelling distinct edges that leave a same node start with distinct letters;
4. the label of each root-to-leaf path in the suffix tree represents a suffix of the input word and the label of each root-to-node path represents a factor;
5. each suffix of w is associated with the label of a (unique) root-to-leaf path in the tree.

Furthermore, an edge links the node spelling ax to the node spelling x for every $a \in \mathcal{A}$ and $x \in \mathcal{A}^*$. Such edges are called *suffix links* and are what allows the tree to be built in time linear with respect to the length of the word.

When the input is a set of words $\mathcal{W} = \{w_1, w_2, \ldots, w_N\}$, a tree called *generalized suffix tree* is used to represent in a compact way all the suffixes of the set of words. A generalized suffix tree is constructed in a way very similar to the suffix tree for a single word. We denote such generalized trees by $\mathcal{GT}_{\mathcal{W}}$ or simply $\mathcal{GT}$ when the input words are clear from the context. Generalized suffix trees have properties similar to those of a suffix tree with word w substituted by the set of words $\mathcal{W}$. In particular, a generalized suffix tree $\mathcal{GT}$ satisfies the fact that every suffix of every word w_i in the set leads to a distinct leaf. When several words share a suffix, the generalized suffix tree must have as many leaves corresponding to the suffix, each associated with a different word. To achieve this property during construction requires simply concatenating to each word w_i a symbol that is not in $\mathcal{A}$ and that is specific to that word.

SPELLER algorithm

We describe in this section the *original* SPELLER algorithm. The algorithm was later extended to allow for more general network expressions (over $\mathcal{P}(\mathcal{A})^+$ and not just $\mathcal{A}^+$) and its performance was improved (see Section 5.3.2).

For ease of exposition, the algorithm is first described as a way of inferring expressions of fixed length that are repeated in a single word. In fact, the length of the expressions output by the algorithm may range over an interval (k_{min}, k_{max}) with possibly $k_{max}) = \infty$. In this last case, the longest expressions output will be those still satisfying the quorum. When $k_{min} = k_{max} = \infty$, only the longest expressions satisfying the quorum are output. It is relatively straightforward to modify the algorithm to treat any of these cases, or to infer expressions common to a set of words, as will be briefly indicated.

SPELLER uses a suffix tree representation of the input word. Actually, it builds (at the same cost) a suffix tree with an additional information attached to each nonleaf node indicating the number of leaves in the subtree rooted at that node. This is also the number of occurrences in w of the factor spelled by the path from the root to the node. Denoting both node and factor by v, the information added to node v is denoted by $\ell(v)$.

Candidate expressions in $\mathcal{A}^+$ are for convenience processed in lexicographical order, starting from the empty word ε. For each candidate expression, say x, all pointers to nodes spelling d-approximate occurrences of x are kept (we say the nodes themselves are (node-)occurrences of x). Let

$$\mathrm{occ}(x, i) = \{y \in F(w) \mid d_H(y, x) = i\}$$

and let

$$\mathrm{occ}_x = \bigcup_{i=1}^{d} \mathrm{occ}(x, i)$$

be the set of occurrences of x. Possibly, some such sets are empty. Let $\ell(x) = \sum_{y \in \mathrm{occ}_x} \ell(y)$. The candidate expression x is processed as long as $\ell(\mathrm{occ}_x) \geq q$. If x has reached the length k, it is output, otherwise its possible extensions are considered. Let xa be its first extension (recall that extensions are attempted in lexicographical order) for $a \in \mathcal{A}$. The occurrences of x belonging to $\mathrm{occ}(x, 0) \cup \mathrm{occ}(x, 1) \cup \mathrm{occ}(x, 2) \ldots \cup \mathrm{occ}(x, d-1)$ are also occurrences of xa. On the other hand, among the occurrences of x that belong to the set $\mathrm{occ}(x, d)$ (their Hamming distance from x is already the maximum allowed), only those followed by a in w may be occurrences of xa. The procedure of extension of a candidate expression is applied recursively. Clearly, if a given candidate expression x no longer satisfies the quorum, it is useless to extend it.

A pseudocode for the algorithm SPELLER is given below. It assumes the suffix tree $\mathcal{T}$ of w has already been built. We define:

$$\mathrm{occ}(x, i)_a = \{y \in \mathrm{occ}(x, i) \mid ya \in F(w)\}$$

(it is the subset of $\mathrm{occ}(x, i)$ followed by the letter a).

```
INITIALIZESPELLER()
1  x ← ε
2  ▷ by convention, the empty word occurs everywhere in w
3  ▷ with Hamming distance 0
4  occ(ε, 0) ← {0, 1, ..., |w| − 1}
5  for i ← 1 to d do
6        occ(ε, i) ← ∅
```

```
SPELLER(x, w, q, k, d)
 1 if ℓ(x) ≥ q then
 2       if |x| = k then
 3             OUTPUT ← OUTPUT ∪ {x, occ_x}
 4       else for a in A do
 5                   for i ← d downto 0 do
 6                         occ(xa, i) ← occ(x, i)_a
 7                         if i ≥ 1 then
 8                               occ(xa, i) ← occ(x, i)_a ∪
                                    (occ(x, i − 1) \ occ(x, i − 1)_a)
 9                   SPELLER(xa, w, q, k, d)
10 return OUTPUT
```

The time complexity of SPELLER is in $O(nV_H(d,k))$ where n is $|w|$ and $V_H(d,k)$ is the size of the set containing all words of length k at Hamming distance d from another of length k. We have that $V_H(d,k) \leq k^d \operatorname{Card}(\mathcal{A})^d$. Therefore, SPELLER is linear in the input size, but possibly exponential with respect to d. It has linear space complexity. When $d = 0$, SPELLER has linear (optimal) time complexity.

When the length of the expressions sought is given as a range of values (k_{min}, k_{max}), the algorithm continues extending candidates as long as they do not reach k_{max}. Any candidate expression x having already reached length k_{min} that satisfies the quorum is output.

In the case where SPELLER is extended to handle expressions in $\mathcal{S} \subseteq \mathcal{P}(\mathcal{A})^+$, $a \in \mathcal{A}$ in line 4 just needs to be replaced by $S \in \mathcal{S}$ while x and xa in lines 6, 8, and 9 are replaced, respectively, by X and XS.

SPELLER can also be applied to infer expressions in $\mathcal{A}^+$ common to a set of words. As mentioned, a *generalized suffix tree* $\mathcal{GT}$ is used in this case to represent all the suffixes of the input words. When we are dealing with N words, it is no longer enough to know the value of $\ell(v)$ for each node v in $\mathcal{GT}$ in order to be able to check whether an expression satisfies the quorum. Indeed, for each node v, we need this time to know not the number of leaves in the subtree of $\mathcal{GT}$ having v as root, but that number for each different word to which the leaves refer.

In order to do that, we associate to each node v in $\mathcal{GT}$ a Boolean array bit_v of size N, that is defined by:

$$bit_v[i] = \begin{cases} 1, & \text{if at least one leaf in the subtree rooted at } x \\ & \text{represents a suffix of } w_i \\ 0, & \text{otherwise} \end{cases}$$

for $1 \leq i \leq N$.

Let $\ell'(x)$ be the total number of cells set to 1 in the Boolean array that results from the OR of bit_v for all nodes v that are occurrences of x in $\mathcal{GT}$.

This corresponds to the number of distinct input words where x occurs. The algorithm then changes only in that the condition to be satisfied now is $\ell'(x) \geq q$.

The time complexity in this case is in $O(nN^2V_H(d,k))$ if n is the length of each input word (assuming for simplicity that they have the same length). The space complexity is $O(nN^2k)$.

MITRA-COUNT **algorithm**

The MITRA-COUNT algorithm proceeds in exactly the same way as SPELLER except that MITRA-COUNT works directly on the input word(s) and not on an index of the word(s) in the form of a (generalized) suffix tree. The time and space cost of building the suffix tree is thus saved. Another advantage of the approach is that the positions of the occurrences of the expressions can be kept naturally ordered as the expressions are recursively extended. This is also a characteristic of earlier algorithms like COMBI and POIVRE which work in essentially the same manner as MITRA-COUNT. On the other hand, the inference step is less efficient both in terms of time (if a factor has multiple copies in the input word(s), it will be processed as many times as it has copies) and of space (for the same reason, factors with multiple copies that are occurrences of an expression will need an equal number of pointers to them).

FOOTPRINTER **algorithm**

FOOTPRINTER has a completely different approach from SPELLER, or from the other approaches that will be described in this chapter. It can address only the problem of inferring expressions common to a set of words. Unique among all the approaches, it also needs as input a phylogenetic tree besides a set of words. In a simplified way, a phylogenetic tree, that we denote by $\mathcal{F}$, is a tree describing the speciation events that have led to the species currently observed, or to those having existed in the past. It is a tree with values attached to the edges whose topology represents the evolutionary relations between the species (current or ancestral) and whose nodes correspond to the species. The value of an edge indicates the evolutionary distance separating the species labelling the nodes at the edge's extremities. Each possible set of factors, one taken from each input word, will be considered, that is, placed at the leaves of the input phylogenetic tree. The parsimony score of the tree is then calculated before deciding whether the set, and the expression at the root of the tree, are a solution of the *substring parsimony problem*. The problem is known to be NP-hard.

Only expressions of a single fixed length k are addressed by FOOTPRINTER. Extension to a range of length values is not straightforward: in practice, the algorithm has to be run again for each different length required for the output expressions.

The algorithm couples a straightforward dynamic programming technique with the use of a table tab_v containing $\mathrm{Card}(\mathcal{A})^k$ entries for each node v of $\mathcal{F}$, including the leaves. All sets of factors are thus treated together. Each entry x (with $x \in \mathcal{A}^k$) in the table corresponds to one possible word of length k to be assigned to node v, and contains the value of the best parsimony score that can be achieved for the subtree rooted at v, if node v is labelled with x. Denote by $C(v)$ the set of children of node v in $\mathcal{F}$. Then table tab_v can be computed for all nodes v of $\mathcal{F}$ starting from the leaves by performing the steps indicated in the following algorithm FOOTPRINTER. The quorum is assumed to be N.

FOOTPRINTER($\mathcal{F}$, $w_1, w_2, \ldots, w_N, k, d$)

for all nodes $v \in \mathcal{F}$ starting from the leaves **do**
  for all $x \in \mathcal{A}^k$ **do**
    if v is a leaf of $\mathcal{F}$ **then**
      ▷ let w_v be the input word placed at leaf v in $\mathcal{F}$
      if x is a factor of w_v **then**
        $tab_v[x] \leftarrow 0$
      else $tab_v[x] \leftarrow +\infty$
    else $tab_v[x] \leftarrow \sum_{u \in C(v)} \min_{y \in \mathcal{A}^k} (tab_u[y] + dist_H(x, y))$
return $\{x \in \mathcal{A}^k \mid tab_{root}[x] \leq d\}$

The algorithm has a structure that resembles the structure of the Fitch algorithm for the so-called *small parsimony problem*. It proceeds from the leaves up to the root looking for the optimum at each level up, and then, once the root has been reached from all leaves, goes down the phylogenetic tree again to recover the values at each internal node and leaf that actually produced all optimal parsimony solutions that are below d.

It is possible to use a quorum lower than N, giving rise to the so-called *substring parsimony problem with losses*. The basic idea is the following. One assumes the evolution time along the edges of the phylogenetic tree is also known, and the quorum is in this case expressed as a minimum total evolutionary time summed over all edges in the subtree containing as leaves the factors that are occurrences of an equivalent expression. An expression may therefore have less than N occurrences, but the occurrences must then span a "wide-enough" evolutionary time, that is, they must concern organisms that are "distant enough" in terms of evolution.

5.2.2. Inference as a clique detection problem

WINNOWER algorithm

The algorithm WINNOWER allows expressions to be inferred that are collections of words in $\mathcal{A}^k$ for a given positive integer k that is the length of the expression. Like FOOTPRINTER (and unlike SPELLER or similar algorithms which do not use an index), there is no efficient way of handling a range of values for the length of the expressions sought.

The method was elaborated to allow expressions common to a set of N input words to be inferred but may easily be adapted to find expressions repeated in a single input word. It can as easily be modified to handle a quorum lower than N although in what follows the method is presented for a quorum of N only.

Given N input words, $w_1, w_2, \ldots, w_N$ of the same length n, an integer d, and a length k, WINNOWER starts by building the graph $\mathcal{G} = (V_1 \cup \cdots \cup V_N, E)$ as indicated in Section 5.1.2. The graph has $O(nN)$ nodes and $O(n^2N)$ edges.

The goal is then to find all cliques of size N in $\mathcal{G}$, which is an NP-complete problem. The idea of WINNOWER is to remove edges that cannot belong to cliques. This makes the graph sparse enough that clique detection is easier to perform.

This is achieved by incrementally eliminating what are called *spurious* edges. An edge is spurious if it does not belong to any extendable clique of a given size where by *extendable* clique of size c is meant a clique contained in *all* other possible cliques of size $c + 1$. By observing that every edge belonging to a clique of size N also belongs to at least $\binom{N-2}{c-2}$ extendable cliques of size c and through a suitable choice of c, it is possible to eliminate spurious edges. This is recursively done as long as possible. At the end, one expects the graph will contain only cliques of size N, or that at least detecting cliques of size N will have become very easy to do in the graph that remains.

The pseudocode is not given in this case as the core ideas are those just described. Care with implementation is required for the efficiency of some essential parts of the algorithm but these are not given in enough detail for us to feel that we can reproduce their essence with perfect fidelity. They are therefore omitted.

The time complexity of WINNOWER is claimed by the authors to be in $O((nN)^{c+1})$ which is the cost of eliminating spurious edges (for $c = 3$, eliminating spurious edges takes on average $O(N^4 n^{2.66})$ time according to them.) If $d = 0$, WINNOWER takes exponential time and is therefore, like FOOTPRINTER, not optimal.

There are interesting instances with critical values of d that cannot be efficiently handled by WINNOWER because too few edges can be eliminated

and the clique detection step must thus be performed in a dense graph. This is the case in what the authors called *the challenge problem*: for instance, for $k = 15$, $d = 4$, and $\text{Card}(\mathcal{A}) = 4$, it is already not feasible to apply WINNOWER to an instance as small as 20 words each of length 600.

MITRA-GRAPH **algorithm**

MITRA-GRAPH is an algorithm that mixes the ideas behind MITRA-COUNT (that is, behind SPELLER) and WINNOWER. It thus works within both the star and clique model. The solutions produced are those that would be obtained with MITRA-COUNT for expressions in $\mathcal{A}^k$ with a distance of d and, originally, a quorum of N. Extending it to expressions of a length covering a range of values, or to a quorum less than N is more straightforward and less costly to do than for WINNOWER.

Like WINNOWER, MITRA-GRAPH builds a graph and looks for cliques of size N in it. The big difference is that the graph depends now at each step on the candidate expression currently considered. The graph is thus denoted by $\mathcal{G} = (x, V_1 \cup \cdots \cup V_N, E)$, or $\mathcal{G} = (x, V, E)$ for short. The set of nodes of $\mathcal{G}$ are defined as in WINNOWER. It is in the set of edges that the two graphs differ. For each node v_i, set $v_i = p_i s_i$ with $|p_i| = |x|$ (and $|s_i| = k - |x|$). In MITRA-GRAPH, there is an edge between v_i and v_j if and only if the three inequalities $dist_H(x, p_i) \leq d$, $dist_H(x, p_j) \leq d$, and $dist_H(x, p_i) + dist_H(x, p_j) + dist_H(s_i, s_j) \leq 2d$ hold. The condition of existence of an edge is therefore stronger with MITRA-GRAPH than with WINNOWER. Finding cliques in this graph is also much easier to do than in the graph used by WINNOWER (it basically eliminates all edges that enter nodes with degree less than $N - 1$), while the pruning ideas of both MITRA-COUNT (when an expression does not satisfy the quorum any longer) and WINNOWER allow this in theory to be a more efficient approach than MITRA-COUNT alone.

The algorithm presents an additional cost due to the fact that the graph has to be updated continuously as viable expressions are recursively explored (in lexicographic order as in MITRA-COUNT). The key idea in this case comes from the observation that once expression xy with $y \in \mathcal{A}^+$, $x \in \mathcal{A}^*$ has been treated, either expression xya with $a \in \mathcal{A}$ will be considered or, if xy did not satisfy the quorum and $y[1], \ldots, y[|y| - 1]$ were all equal to the last letter in the alphabet, it is expression xb with $b \in \mathcal{A}$ and different from the first letter in y (it will, in fact, be the next letter in the alphabet) that will be considered. From the graph $\mathcal{G}(xy, V, E)$, it is easy to obtain $\mathcal{G}(xb, V, E)$ if the values of $dist_H(x, v_i[0\,..\,(|x| - 1)])$, $dist_H(x, v_j[0\,..\,(|x| - 1)])$, $dist_H(v_i[|x|\,..\,(k - 1)], v_j[|x|\,..\,(k - 1)])$ are kept for each edge (v_i, v_j).

As for WINNOWER and for the same reason (not enough detail is presented in the literature indicating how the algorithm is actually implemented), a pseudocode for MITRA-GRAPH is omitted.

5.3. Inferring network expressions with spacers

5.3.1. Mathematical models and related inference problem

In biology, network expressions with spacers are a first approach to model sequences along a molecule, typically DNA, that function in a cooperative way in the sense that they need to simultaneously bind a same or different molecular complex so that a given biological process may be initiated. In the case of so-called "higher" organisms, the sites may even come in big clusters. The relative positions along the molecule of the sites that are inside a cluster are in general not random, because they are recognized either by the same complex and cannot therefore stand too much apart, or by different complexes that interact between them. In this last case, the distances between sites along the molecule may be longer but are often quite constrained. Finally, not all positions within a site are equally important for the binding to happen. In particular for binding sites in proteins where recognition is strongly connected to the 3D structure of the molecule, even a single binding site (single in the sense that it binds a unique site in the other molecule) may concern a sequence of noncontiguous positions at variable distances one from another that correspond to amino acids close in 3D space.

Given an alphabet $\mathcal{A}$, a network expression X with spacers is an ordered sequence of simple network expressions $X_1, \ldots, X_p$ with $X_1, \ldots, X_p \in \mathcal{P}(\mathcal{A})^+$ and $p \geq 2$.

The expression X is said to appear exactly in a word w if there exist factors $u_1, \ldots, u_p$ of w such that $w = t_0 u_1 t_1 \cdots t_{p-1} u_p t_p$ with $t_0, \ldots, t_p \in \mathcal{A}^*$ and $u_i \in X_i$ for $1 \leq i \leq p$. Given $d = (d_1, \ldots, d_p)$ nonnegative integers, X is said to appear d-approximately in w if $w = t_0 u_1 t_1 \cdots t_{p-1} u_p t_p$ with $t_0, \ldots, t_p \in \mathcal{A}^*$ and, for all $i \in [1, p]$, there exists $v_i \in X_i$ such that $dist_H(u_i, v_i) \leq d_i$.

Finally, given a network expression X with constrained spacers, that is, given a sequence of simple network expressions $X_1, \ldots, X_p$, positive integers $d_1, \ldots, d_p$, and intervals $[min_1, max_1], \ldots, [min_{p-1}, max_{p-1}]$ with $min_i \leq max_i$ nonnegative integers, X is said to appear approximately in a word w if $w = t_0 u_1 t_1 \cdots t_{p-1} u_p t_p$ with $t_0, \ldots, t_p \in \mathcal{A}^*$, u_i a d_i-approximate occurrence of X_i for all $i \in [1, p]$ and $|t_j| \in [min_j, max_j]$ for all $j \in [1, p-1]$. The case of intervals containing negative values may also be considered but has not been treated in the literature.

From now on, network expressions X with constrained spacers and composed of p simple network expressions $X_1, \ldots, X_p$ separated by distances within the intervals $[min_1, max_1], \ldots, [min_{p-1}, max_{p-1}]$ will be denoted by $X = X_1\ [min_1, max_1]\ X_2, \ldots, X_{p-1}\ [min_{p-1}, max_{p-1}]X_p$. Network expressions with unconstrained spacers and composed of p simple network expressions $X_1, \ldots, X_p$ will be denoted by $X = X_1 * \cdots * X_p$.

5.3.2. Algorithms

Inferring network expressions with constrained spacers

MITRA-DYAD **algorithm.** MITRA-DYAD infers network expressions with constrained spacers only for the case $p = 2$, that is expressions of the type $X = X_1[min, max]X_2$. The reason is that the inference is performed in a MITRA-GRAPH but containing $O(max - min + 1)$ times more nodes and potentially $O((max - min + 1)^2)$ more edges. Indeed, supposing $|X_1| = |X_2| = k$, each factor u of length k of the input words $w_1, \ldots, w_N$, which corresponded to a node in the original graph, now gives rise to $O(max - min + 1)$ nodes, each one corresponding to the factor u followed by the factor v starting i positions after the end of u, when such a position exists, for i between min and max. Nodes are then linked under the same conditions as for MITRA-GRAPH; in particular, the existence of an edge between two nodes remains dependent on the expression $X = X_1[min, max]X_2$ that is being currently considered. Once this graph is built, MITRA-DYAD runs MITRA-GRAPH on it. The way the graph is built ensures that the solutions found in this way correspond to the required network expressions with constrained spacers.

SMILE **algorithm.** There are in fact two versions of the SMILE algorithm. Both versions call the SPELLER algorithm given in Section 5.2.1 as an internal subroutine. The basic algorithm for a single input word is shown below. It is straightforward to adapt it to the case of N input words. For ease of exposition, we assume that, in the expression $X = X_1[min_1, max_1]X_2, \ldots, X_{p-1}[min_{p-1}, max_{p-1}]X_p$, all expressions X_i have the same length k and maximum number of differences allowed d. The way that the search space is considered makes the main difference between the two versions of SMILE. It is worth observing that besides being able to handle a different distance d for each simple expression in X, SMILE can handle a global distance, something MITRA-DYAD cannot do.

The SMILE algorithm shown below assumes that $p = 2$ and that the suffix tree $\mathcal{T}$ of w has been previously built. The notations X_1 and X_2 stand for candidate motifs for, respectively, the first and second expressions in the network expression containing the spacer that is being searched for.

SMILE$(w, q, k, d, (min_1, max_1), \ldots, (min_p, max_p))$

```
1 for i ← 1 to p − 1 do
2     for each solution of SPELLER(X_1, w, q, k, d) do
3         consider only the search space of all factors of w
4         that start from min_i to max_i positions after
5         occurrences of X_1 in w
6 return SPELLER(X_2, w, q, k, d)
```

The extension of SMILE to the case where $p > 2$ is straightforward. The difference between the two versions of the SMILE algorithm is in how lines 3 to 5 are dealt with. We explain it in the simple case where $p = 2$ and $|X_1| = |X_2| = k$. Generalization to different lengths (or range of lengths) for each simple expression in X, or to a general p, is straightforward for the first version and more elaborate for the second. Details may be found in the literature indicated in the notes at the end of the chapter.

The first version of SMILE proceeds as follows. For each expression X_1 of length k satisfying the quorum that is obtained, together with its set of node-occurrences in $\mathcal{T}$, that we denote by occ_{X_1}, all simple expressions X_2 are sought. The search starts (using SPELLER) with the expression $X_2 = \varepsilon$ and occ_{X_a} the set of words v which have an ancestor u in occ_{X_1} with $min \leq level(v) - level(u) \leq max$, where $level(v)$ indicates the length of the label of the path from the root to node v in $\mathcal{T}$. From a node-occurrence u in occ_{X_1}, a jump is therefore made in $\mathcal{T}$ to all potential start node-occurrences v of X_2. These nodes are the *min* to *max*-generation descendants of u in $\mathcal{T}$.

The second version of SMILE initially proceeds like the first. For each simple expression X_1 inferred, and for each node-occurrence u of X_1 considered in turn, a jump is made in $\mathcal{T}$ down to the descendants of u located at lower levels. This time, however, the algorithm just passes through the nodes at these lower levels, grabs some information the nodes contain and jumps back up to level k again. The information grabbed in passing is used to temporarily and partially modify $\mathcal{T}$ and start, *from the root of* $\mathcal{T}$, the inference of all possible companions X_2 for X_1 that are located at the required distance (min, max). Once this operation has ended, the part of $\mathcal{T}$ that was modified is restored to its previous state. The inference of another simple expression X_1 then follows. The whole process unwinds in a recursive way until all expressions X satisfying the initial conditions are inferred.

More precisely, the operation between the spelling of X_1 and X_2 locally changes $\mathcal{T}$ up to level k into a tree $\mathcal{T}'$ that contains only the prefixes of length k of suffixes of w starting at a position between *min* and *max* from the end position in w of an occurrence of X_1. Tree $\mathcal{T}'$ is, in a sense, the union of all the subtrees t of depth at most k rooted at nodes that represent

start occurrences of a potential companion X_2 for X_1. SPELLER can then be applied directly to $\mathcal{T}'$. The information that is grabbed in passing is the one required to modify $\mathcal{T}$ into $\mathcal{T}'$: it corresponds to the Boolean arrays indicating to which factors of w belong the leaves of all potential end node-occurrences of companions for X_1 in the tree.

The complexity of the first version of SMILE for a single input word is $O(n + n_{2k+max}V_H^2(d,k))$ where n_{2k+max} is the number of nodes at level $2k + max$ in the suffix tree. Its space complexity is $O(n(2k + max))$.

The complexity of the second version of SMILE for a single input word is $O(n + \min\{n_k^2, n_{2k+max}\}V_H^2(d,k) + n_{2k+max}V_H(d,k)))$ and its space complexity $O(n(2k + max) + n_k)$.

Inferring network expressions with unconstrained spacers

SMILE algorithm revisited. Extensions of the SMILE algorithm also enable flexible spacers to be dealt with.

The first extension concerns what is called "meta-differences". Given a nonnegative integer D, a network expression $X = X_1[min_1, max_1]X_2, \ldots, X_{p-1}\ [min_{p-1}, max_{p-1}]X_p$ is said to appear exactly in a word w if there exist factors $u_{j_1}, \ldots, u_{j_q}$ of w such that $p - D \leq q \leq p$ and $w = t_0 u_{j_1} t_1 \cdots t_{q-1} u_{j_q} t_q$, with $t_0, \ldots, t_q \in \mathcal{A}^*$, $1 \leq j_1 < \cdots < j_q \leq p$ and $u_{j_i} \in X_{j_i}$ for $i = 1, \ldots, q$. An equivalent definition may be derived for approximate occurrences of X.

The second extension allows SMILE to handle restricted intervals of distances between the simple network expressions $X_1, \ldots, X_p$, exploring in a same run a wide range of possibilities for the middle value of the interval. The expressions that may be inferred in this case are of the type $X = X_1[m_1 \pm \varepsilon_1]X_2, \ldots, X_{p-1}[m_{p-1} \pm \varepsilon_{p-1}]X_p$ where, for $1 \leq i < p$, ε_i is a nonnegative integer and $m_i \in [Min_i, Max_i]$ with $Max_i - Min_i$ as large as desirable.

PRATT algorithm. Trying to infer network expressions with completely unconstrained spacers would in most situations lead to trivial solutions besides being a computationally harder problem. PRATT therefore imposes some constraints on the amount and distribution of do not care symbols that are allowed. The expressions treated may however be more flexible than the constrained spacers of SMILE as presented in Section 5.3.2. We shall see in a moment the extensions of SMILE which enable spacers that are as unconstrained to be dealt with as in PRATT, although in a different way.

The constraints that PRATT puts on the spacers are specified as input parameters. Some of the main ones are:

1. a maximum number of spacer regions, that is of regions that are composed of a contiguous sequence of do not care symbols;
2. a maximum length for spacer regions;
3. a maximum number of overall do not care in the expressions sought;
4. a maximum length of the network expression.

Other possible constraints are omitted for the sake of simplicity.

SPELLER takes in general as input N words, that is, it infers common network expressions, but it can easily be modified to treat the case of a single input word. It works basically like SPELLER for expressions in $\mathcal{S} \subseteq \mathcal{P}(\mathcal{A})^+$. Unlike SPELLER, PRATT does not use a suffix tree representation of the input word(s) but a simple queue or file data structure like COMBI or POIVRE. The do not care symbol is treated in a way similar to another element of $\mathcal{S}$, with counters enabling whatever spacer constraint was given as input to be checked.

The version of SMILE that allows intervals for the distances between single network expressions results in performances analogous to those of PRATT as far as spacers are concerned, although in a slightly different way. SMILE can be more flexible and it further allows for differences in the inference process.

5.4. Related issues

5.4.1. The concept of basis

Given some input word(s), the number of even simpler expressions $X \in \mathcal{A}(\mathcal{A} \cup \bullet)^*\mathcal{A}$ can be exponential with the length of the input, so that it is infeasible to list all of them along with their occurrences in the word(s). Fixing the Hamming distance to 0 or using a high quorum does not avoid the explosive growth in the number of such expressions. Several researchers are working to alleviate this drawback.

Among the many methods proposed to select expressions, one can single out those based on the notion of *maximality* or *specificity*. We assume $d = 0$. Since the expressions may contain do not cares, approximate occurrences are in a certain sense still allowed. Informally, an expression X in $\mathcal{A}(\mathcal{A} \cup \bullet)^*\mathcal{A}$ is maximal if it cannot be extended to the left or to the right by adding further symbols and/or if none of its do not care symbols can be replaced by an alphabet letter, without losing any occurrences. In other words, specifying more a maximal expression causes a loss of information, while this is not true for nonmaximal expressions. While the notion of maximality reduces significantly the number of expressions, their number may still be exponential.

A significant step in reducing the number of maximal expressions is the introduction of the notion of *basis*. Informally speaking, a basis is a set of (maximal) expressions that can generate all the (other) maximal expressions by simple mechanical rules. The maximal expressions in the basis are representative of the information content of the words in that they can generate all the other repeated or common expressions. A notion of basis called the set of *tiling motifs* was introduced. It has size linear in the length n of the input word(s) and it is able to generate the repeated expressions (possibly exponential in number) that appear at least twice with do not care symbols in such input over an alphabet $\mathcal{A}$. This basis has some interesting features such as being (a) a subset of previously defined bases; (b) truly linear as its expressions are less than n in number and appear in the word for a total of $2n$ times at most; (c) symmetric as the basis of the reversed word is the reverse of the basis; (d) computable in polynomial time, namely in $O(n^2 \log n \log \mathrm{Card}(\mathcal{A}))$ time. As an example, the basis of tiling motifs for repeats in the word $w = \mathtt{ATATACTATCAT}$ contains three elements, namely $x_1 = \mathtt{ATA{\bullet}{\bullet}{\bullet}TAT}$, $x_2 = \mathtt{ATAT{\bullet}{\bullet}T}$, and $x_3 = \mathtt{TATA{\bullet}{\bullet}AT}$ that are able to generate (through a suitable operation that also takes into account the positions where the motifs in the basis occur) all other repeated motifs such as TAT, TA, AT, $\mathtt{ATA{\bullet}{\bullet}{\bullet}T}$, etc. that appear at least twice in w. For instance, the motif $\mathtt{ATA{\bullet}{\bullet}{\bullet}T}$ can be obtained by the overlap of the occurrences of x_1 and of x_2 at position 0.

A more general and flexible framework is required for repeated or common expressions when $d > 0$ and the notion of a basis may perhaps not be extended in this case. Some fuzzy form of clustering should then be considered.

5.4.2. Inferring tandem network expressions

Problem definition

Tandem arrays (called *tandem repeats* when there are only two units) are approximate powers (squares) of a word, that is, a sequence of approximate repeats that appear adjacent in a word. The inference of tandem arrays may proceed in much the same way as for simple expressions that are repeated a number of times in a word (using for instance SPELLER or MITRACOUNT). Checking that the expression appears *tandemly* repeated can then be done a posteriori. This however can be a very inefficient approach as many expressions will be generated whose occurrences have no chance of forming a tandem array. It is therefore more interesting to develop a method that allows the tandem condition of a repeat to be checked as it proceeds with the inference, that is, simultaneously with it. The use of a suffix tree

is not interesting when approximate matches are sought because a suffix tree does not allow the positions of the occurrences to be kept ordered for easy processing of the tandem condition. An approach like the one adopted by MITRA-COUNT, that was also used earlier in COMBI or POIVRE, is the most appropriate in this case.

Before sketching the main ideas of the algorithm, called SATELLITE, we need to introduce the more complex models required by tandem arrays. There are in fact two definitions related to a tandem array model, one called *prefix model* and the other *consensus model*. This latter concerns tandem array models strictly speaking while prefix models are in fact models for approximately periodic repeats that are not necessarily (yet) tandem. They correspond to the prefixes of a consensus model.

Formally, a *prefix model* of a tandem array is a word $x \in \mathcal{A}^+$ (x could also belong to $\mathcal{P}(\mathcal{A})^+$) that approximately matches a *train of wagons*. A *wagon* of x is a factor u in w such that $dist_E(x, u) \leq d$ for d a nonnegative integer (observe that in this case, it is the edit distance that has been considered). A *train* of a prefix model x is a collection of wagons $u_1, u_2, \ldots, u_p$ ordered by their starting positions in w and satisfying the following properties:

- (P_1) $p \geq q$ where q is again a quorum indicating this time the minimum number of units the sought tandem arrays must have;
- (P_2) $left_{u_{i+1}} - left_{u_i} \in [min\_period, max\_period]$ is the position of the left end of wagon u in w and *min_period*, *max_period* are the minimum and maximum period of the repeat.

A consensus model must further satisfy the following property:

- (P_3) $left_{u_{i+1}} - right_{u_i} = 0$

where $right_u$ is the position of the right end of wagon u. The property checks that the occurrences of consensus models are indeed tandem. This is verified only when $|x| \in [min\_period, max\_period]$, that is when the length of the repeat has reached the value specified as input.

The tandem array inference problem is then the following.

INFERENCE OF TANDEM ARRAY PROBLEM

INPUT: A word $w \in \mathcal{A}^+$, a quorum q, an edit distance d, and a minimum and maximum period *min_period* and *max_period*.
OUTPUT: All expressions $x \in \mathcal{A}^+$ that are consensus models for tandem arrays (that is, properties (P_1), (P_2), and (P_3) are satisfied).

SATELLITE **algorithm**

Expressions for tandem arrays are inferred by increasing length. The algorithm keeps track of individual wagons, and at each step determines, on the

fly, if they can be combined into at least one train (observe that a wagon can belong to more than one train). The latter corresponds to checking, for each wagon, whether it belongs to at least one set of wagons satisfying these properties (P_1) and (P_2/P_3).

For each expression x that is a prefix model for a tandem array, a list of the wagons of x that belong to at least one train of x is kept. When the expression x is extended into the expression $x' = xa$, two tasks must be performed:

1. determine which wagons of x can be extended to become wagons of x';
2. among these newly-determined wagons of x', keep only those that belong to at least a train of x'. This requires effectively assembling wagons into trains.

The trains do not need to be enumerated in the second step, it sufficer to determine whether if a wagon is part of a train. This allows an extension step to be performed in time linear with the length of the input word.

Consider the directed graph $G = (V, E)$ where V is the set of all wagons of x and there is an edge from wagon u to wagon v if $left_v - left_u \in [min\_period, max\_period]$. A wagon u is then part of a train if it is in a path of length q or more in G. Determining this is quite simple as the graph is clearly acyclic.

For each expression x of length between *min_period* and *max_period*, it remains to check whether x satisfies the properties of a consensus model for a tandem array. Consider now a directed bipartite graph $G_x = (L_x \cup R_x, E)$ whose vertices are the positions at which, respectively, the left and right ends of wagons of x occur. Edges $i \to j$ with $i \in L_x$, $j \in R_x$ are *wagon edges* and edges $j \to i$ with $i \in L_x$, $j \in R_x$ are *gap edges*. There is a wagon edge $i \to j$ if and only if $w[i \,..\, j-1]$ is a wagon, and there is a gap edge $j \to i$ if and only if $i = j$. Thus, an edge sequence $i \to j \to k$ occurs in G_x if and only if there are wagons u and v such that $u = w_i w_{i+1} \cdots w_{j-1}$, $left_v = k$, and $left_v - right_u = 0$. It follows that a position/node which is on a path of length $2q$ or more is part of a train satisfying properties (P_1), (P_2), and (P_3). Such a position is called a *final position* or *final node*. Let G'_x be the graph induced by the set, F_x, of all final nodes. If G'_x is nonempty, then x is a consensus model for a tandem array having the characteristics specified in the input.

The complexity of SATELLITE is $O(n\ max\_period\ M_E(d, k))$ where n is, as before, $|w|$ and $M_E(d, k)$ is the size of the set containing all words of length k at edit distance d from another word of length k. This is actually the complexity of SPELLER multiplied by the term *max_period* because of the need to check for the tandem condition. An extended version of SATELLITE allows tandem arrays that may miss a period to be dealt with, meaning that

the repeat may contain some units that have accumulated more differences than allowed. Such units are called *bad wagons*. A number of them may be authorized in a train.

5.5. Open problems

5.5.1. Inference of network expressions using edit distance

In theory, all algorithms presented in this chapter may be modified to handle edit instead of Hamming distance. Indeed, edit distance is already an integral part of POIVRE (and of SATELLITE). Thus MITRA-COUNT, which behaves much as POIVRE, can easily be extended to use an edit distance. The same is true of SPELLER and such a modification was suggested and quickly sketched by the authors. A more recent approach using a suffix tree such as SPELLER introduces what appears to be an algorithm producing a different solution from SPELLER given the same instance.

There has also been a theoretical discussion on how to introduce edit distance into FOOTPRINTER, which includes the time complexity that the resulting algorithm would have. However, FOOTPRINTER is not suitable for dealing with expressions or occurrences of variable length which appear when insertions and deletions are allowed. The reason comes from the data structure used (the table at each node of the tree).

WINNOWER could also theoretically handle an edit distance, but the number of edges in the graph would grow as would the number of spurious edges. MITRA-GRAPH would have the same type of problem but the filtering of spurious edges is easier to perform and therefore the algorithm might be able to handle the situation much better than WINNOWER.

Finally, the first version of SMILE is, like SPELLER, easily modifiable to handle an edit distance. Although theoretically not impossible, introducing such distance into the second version of SMILE might be more tricky.

In all cases, it is worth exploring more compact ways of representing the occurrences of an expression once insertions and deletions are allowed. One possible way extends ideas for pattern matching in a long text with edit distance.

5.5.2. Minimal covering set

The concept of *minimal covering set* of expressions may enable two difficulties encountered to be addressed by currently existing combinatorial algorithms for network expression inference in a set of words. These difficulties are, first how to fix a priori the quorum, and second (an even harder problem) how to efficiently identify weak and rare expressions? To

solve the second problem one can increase the value of d while simultaneously decreasing the value of q. However, this may lead to a huge number of solutions, many of which are uninteresting. A minimal covering set extends the concept of individual expressions, with or without spacers, to that of a family of expressions which "completely explains" a set of words. In a more precise way, the problem could be expressed in the following (informal) way. Given a set of input words, one must find a minimal set of $r \geq 1$ expression(s) (the value of r is unknown at start) such that:

- each expression has an occurrence in at least one input word;
- distinct expressions among the r may have occurrences in the same input word but the number of times this may happen is smaller than a threshold value t (possibly t may be 0: there is no "word overlap" of the expressions);
- all words are covered by (at least) one expression in the family (strictly one in case the threshold t is 0).

Notes

The term *network expression* to denote repeated regular expressions without the Kleene closure was introduced for the first time by Mehldau and Myers (1993).

The literature on grammatical inference is large. Three papers have been influential on the theory of learning grammars. The first by Gold (1967) introduced the notion of "language identification in the limit", the second by Wharton (1974) relaxed the condition of an exact identification by allowing for various descriptions of the correct solution, while the third by Valiant (1984) relaxed such a condition by allowing for a solution to be only approximately correct. The earliest and main expository of inference problems for regular grammars is Angluin (1982, 1987).

The definitions and statements concerning the star model have been adopted in several exact algorithms, namely COMBI (Sagot and Viari 1996), POIVRE (Sagot, Viari, and Soldano 1997), SPELLER (Sagot 1998), SMILE (Marsan and Sagot 2000b, 2001), PRATT (Jonassen, Collins, and Higgins 1995), and MITRA-COUNT (Eskin and Pevzner 2002). The uniform property variant of the star model has been introduced by Sagot (1996) and a similar idea in a previous paper (Sagot, Soldano, and Viari 1995).

The closest substring problem has been defined and proved to be NP-complete in (Fellows, Gramm, and Niedermeier 2002). It remains an open problem whether it is parameter-tractable for a constant size alphabet when either d alone or d and N are fixed (Fellows *et al*. 2002).

The definitions of the *substring parsimony problem* and the proof of its NP-hardness are given in Blanchette, Schwikowski, and Tompa (2000). The *small parsimony problem* was introduced in (Fitch 1975) and the *substring parsimony problem with losses* in Blanchette (2001); Blanchette, Schwikowski, and Tompa (2002). The FOOTPRINTER algorithm was presented and analysed in Blanchette *et al.* (2000); Blanchette (2001); Blanchette and Tompa (2002); Blanchette *et al.* (2002). The time complexity of FOOTPRINTER is $O(Nk\,\mathrm{Card}(\mathcal{A})^k + nNk)$, where n is the length of each input word (assuming they have the same length). The highest term was in fact $Nk\,\mathrm{Card}(\mathcal{A})^k$ in a first paper (Blanchette *et al.* 2000). This came from the fact that the computation of the Hamming distance between two words of length k (which takes $O(k)$ time) is done for each of the $O(N)$ edges in $\mathcal{F}$, each of the $O(\mathrm{Card}(\mathcal{A})^k)$ possible values for x, and each of the $O(\mathrm{Card}(\mathcal{A})^k)$ possible values for y. In Blanchette (2001), an improvement of the original algorithm described in Blanchette *et al.* (2000) was introduced which enabled the exponent k instead of $2k$ to be obtained. The improvement is achieved by means of an auxiliary table for each edge in $\mathcal{F}$. Details may be found in Blanchette (2001). FOOTPRINTER is thus linear with the size of the input words but exponential with the length k of the expressions sought. If $d = 0$, FOOTPRINTER still takes exponential time and is therefore not optimal. The algorithm WINNOWER was described in Pevzner and Sze (2000). One must observe that if a quorum lower than N is used, the size of the cliques sought is the only thing that changes in WINNOWER. In practice, however, the smaller the quorum, the less spurious edges there will be that can be safely eliminated. Winnower's descendant was presented in Eskin and Pevzner (2002); Eskin, Gelfand, and Pevzner (2003). The two algorithms that are similar to WINNOWER and MITRA-GRAPH, namely KMRC and POIVRE, appeared in Sagot, Viari, Pothier, and Soldano (1995); Soldano, Viari, and Champesme (1995).

Information concerning the Profile data base can be found in Buhler and Tompa (2001).

Some examples of the use of relative entropy for the evaluation of network expressions can be found in Vanet, Marsan, and Sagot (1999); Pavesi, Mauri, and Pesole (2001b).

The algorithm SPELLER was introduced in Sagot (1998), while the two variants it inspired, MITRA-COUNT and WEEDER, were described in, respectively, Eskin and Pevzner (2002) and Pavesi, Mauri, and Pesole (2001a). Detailed information about the generalized suffix tree data structure can be found in Bieganski, Riedl, Carlis, and Retzel (1994); Hui (1992). A recent approach using a suffix tree as in SPELLER but working with the edit distance is given in Adebiyi, Jiang, and Kaufmann (2001); Adebiyi and Kaufmann (2002). The approach seems to produce a different solution from

the one that would result from an application of an extension of SPELLER enabling the edit distance to be worked with.

Concerning algorithms for inferring network expressions with constrained spacers, the various versions of the SMILE algorithm are described in Marsan and Sagot (2000b, 2001), and MITRA-DYAD is presented in Eskin and Pevzner (2002); Eskin *et al*. (2003).

For the case of unconstrained spacers, PRATT is introduced in Brazma, Jonassen, Vilo, and Ukkonen (1998c, 1998b); Brazma, Jonassen, Eidhammer, and Gilbert (1998a); Jonassen *et al*. (1995). A few years after PRATT was conceived, an algorithm that is roughly equivalent to PRATT in terms of its output was elaborated which uses a lazy implementation of the suffix tree (Giegerich, Kurtz, and Stoye 1999) to represent the patterns as these are produced. The lazy suffix tree construction as adapted by the authors to their needs takes quadratic time but is claimed to be efficient in most practical situations (Brazma *et al*. 1998c).

The notion of basis of repeated motifs was introduced in Parida, Rigoutsos, Floratos, Platt, and Gao (2000); Parida, Rigoutsos, and Platt (2001). The more recent notion of tiling motifs was described in Pisanti, Crochemore, Grossi, and Sagot (2003).

Finally, the SATELLITE algorithm for inferring tandem network expressions can be found in Sagot and Myers (1998).

Readers interested in approaches to the inference, in biological applications, of simple network expressions or of network expressions with spacers using heuristics or statistical methods may consult Durbin, Eddy, Krogh, and Mitchison (1998); Pevzner (2000); Waterman (1995). Machine learning techniques have also long been used to infer patterns or grammars. References to some of these techniques as they apply to biology may be found in Baldi and Brunak (1998).