Eötvös Loránd University

Faculty of Science

Bálint Márk Vásárhelyi

Mathematical methods in DNA sequence analysis

BsC Diploma Thesis

Supervisor:

Kristóf Bérczi
Department of Operations Research

Budapest, 2011

Contents

Introduction

Nowadays, one of the most promising fields of mathematics is biomathematics. There are several biological problems that can be treated with mathematical methods, and it seems to be very useful for both mathematics and biologics to combine the results of the two disciplines. One of the most important problems is genetic mapping and sequence analysis of DNA. There are several databases, the most well-known is GenBank, which was created by the Los Alamos National Labs, but now it is maintained by the National Center for Biotechnology Information. According to [14], the size of stored DNA sequences was increased by about 75 % each year in the 1990s. Today, this number is even more large, as different methods were published since that time. In this thesis, several methods are shown used for mapping and sequence analysis.

In Chapter 1, several ways of mapping are introduced. The main topic of this part is STS-content mapping, which is a wide-spread method of sequence analysis. We also consider genetic mappings.

Another approach of STS-content mapping is shown in Chapter 2. The aim of this method is to find a layout of the clones, requiring the least space possible.

A special mapping problem is acquainted in Chapter 3. Sometimes, not all the relations of mapping points are known, just a few of them. Here, only some triplets are given with their central elements, and we present a method for finding an order of the elements that satisfies as more constraints as possible.

Comparison of two or more sequences is also of interest. A number of methods for comparison is shown in Chapter 4. One may be curious about knowing if a short sample is contained by a longer sequence, or we can compute the edit distance of two strings.

I say thanks to my supervisor, Kristóf Bérczi for his devoted professional help. I thank to András Frank for making me available the book *Algorithms on Strings, Trees and Sequences* of Dan Gusfield.

SDG

Chapter 1

Physical and genetic mappings

In general, we distinguish **genetic maps** and **physical maps** [6]. Physical maps are easier to be handled by mathematical methods, as here a DNA sequence can be handled as a string.

Physical mappings establish on the true physical location of markers or known patterns, such as microsatellites (microsatellites are repeating short substrings, usually two bases, for example $ACACACACAC$ is a microsatellite). Usually the distance of two markers is the number of nucleotides between them. With this map one can roughly place a gene on the map, we can notice a deletion or an insertion. For example, the tumor suppressor gene of retinoblastoma was firstly localized by observing a deletion on the chromosome 13 [13]. The aim of physical mapping is to locate the interesting genes on their base-pair location on a chromosome.

On the other side, genetic mappings are based on observing the degree of recombination. This notion represent the relative frequency of cross-overs in a specified section of meiosis, and it is in connection with linked inheritance of genes at two loci on the DNA. (See Figure 1.1.) It is supposed that the higher the frequency of

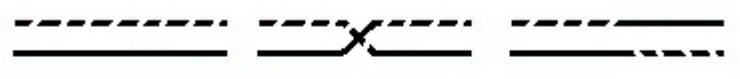

Figure 1.1: Cross-over between two double-chained DNA

coinheritance, the closer the two loci are. Genetic mappings are more beneficial in genetics, because they reveal more information about the alleles on the DNA than physical mapping. The unit of genetic mapping is called **centimorgan**, which is usually referred to the distance of two alleles where the degree of recombination is 0.01. More precisely, the distance is $d = \frac{1}{2}\ln(1 - R)$, where R is the degree of recombination. The problem of genetic mappings is that the collection of data is more difficult, and centimorgan is not an absolute unit, but it is characteristic of species.

Now, we will study physical mappings.

1.1 Physical mapping

1.1.1 Clone libraries

A very important mapping strategy is the **STS-content mapping**. STS stands for sequence-tagged-site, and it is a 200-300 nucleotides long DNA substring, whose left and right 20-30 bases occur only once in the entire genome. One of the first goals of the Human Genome Project was to find a set of STSs such that any 100,000 nucleotides long DNA substring contains at least one of them. [5]

We say that a **clone library** is a set of short DNA fragments or **clones** that can overlap and that cover the whole DNA. The order of the clones are unknown. One aim is to build a **ordered clone library**.

The goal of STS-content mapping is determining the order of STSs and building the ordered clone library. First, the method determines which clone contains which STSs. It can be implemented by hybridization or by PCR technology. The second step is reconstructing the order of the STSs and placing the clones on a physical map. Two basic ideas are used. The first is that the distance between STS_i and STS_j is inversely related to the number of clones containing both STS_i and STS_j. The second is that if STS_i occurs in both clones $clone_k$ and $clone_l$, then $clone_k$ and $clone_l$ must overlap at STS_i (see Figure 1.2).

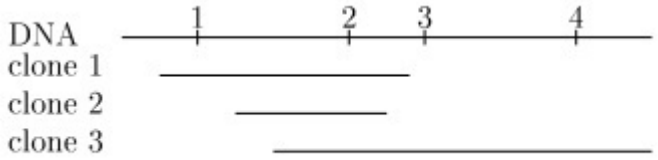

Figure 1.2: Input data of STS mapping

	1	2	3	4
1	1	1	0	0
2	0	1	0	0
3	0	1	1	1

Table 1.1: The input matrix constructed from data in Figure 1.2

From the data of the PCR, an input matrix can be constructed (see Table 1.1). The intersection of row i and column j would be 1 if $clone_i$ contains STS_j, otherwise 0. The aim is to permute this matrix so that in each row, the ones are not separated by zeros. Booth and Lueker [2] shows an algorithm for finding such a permutation or proving that there is no such a permutation.

1.1.2 Errors of STS-mapping

There are three important systematic errors of STS-content mapping, namely, the **false negative report**, the **false positive report** and the **chimeric clones**. False negative report is when it is reported that the clone does not contain the STS, but actually it does. Respectively, false positive report is when it is reported that an STS occurs in a clone, despite the fact that it does not.

Chimeric clones are a more complicated type of error. Sometimes, two different fragments of the original DNA join to each other, and a new DNA clone is created. However, the location of the two fragments can be widely differing, which means a great problem in STS-mapping. Existence of chimeric clones clearly makes the mapping more difficult, and unfortunately the more fragments we have the more chimeric clones will be formed.

1.2 Genetic mapping

Genetic maps are based on genetic recombination. Genetic recombination is a new allele combination which is different from the parental one and it occurs when there is an odd number of crossing overs between the two homologous chromosomes in the section between two interested alelles. The ideal crossing over is pointwise, autonomous and it occurs with a certain probability at each point [11].

One of the mapping functions is the Haldane function, which points at a relation between the genetic linkage R (which is the relative frequency of odd number of crossing overs) and the serial distance d between two alleles.

$$R = \left(1 - e^{-2d}\right)$$

The distance is given in centimorgans. From the Taylor series of e^x the distance of two alleles is one centimorgan if the relative frequency of crossing over between the two alleles is approximately 1%. Of course, the Haldane function can be generalized for more than two mapping points.

Chapter 2

The tightest layout

2.1 Tightest layout of clones

As we have seen in Chapter 1, in physical mappings we are given some clones (substrings) from a DNA and some probes. It is known that which probes contain which clones, and we have to show an arrangement of the probes. Karp et al. [1] developed a method for this problem.

Let $\mathcal{P}$ be a set of probes and $\mathcal{C}$ be a set of clones. $|\mathcal{C}|$ is denoted by n. For each clone $c \in \mathcal{C}$ let P_c be the set of the probes containing c. It is possible that a probe occurs at more than one location. The number of times a probe appears is not known.

We say that a mapping of clones in $\mathcal{C}$ and the probe occurrences is a **feasible layout**, if each clone $c \in \mathcal{C}$ is contained at least one copy of each probe in P_c, but no copies of probes in $\mathcal{P} \setminus P_c$. Because a probe may occur at more than one location, a feasible layout can be always shown: place each clone c separated from the others, and place one copy of each required probe on the clone. We say that this is a **primitive feasible layout**. Our aim is to find a tightest layout, which occupies the least space on the real line and in which the fewest probe copies occurs.

Karp et al. [1] made two restriction to solve this problem. Assume that each clone has equal length, and predetermine a permutation of the clones, so choose an input permutation, in which the names of the clones are $1, .., n$.

Lemma 2.1.1. *Let $i < j < k$ be three clones. If there is a $p \in \mathcal{P}$ occurring in clone j, but neither in clone i nor k, then clones i and k are disjoint.*

Proof. Suppose that i and k overlap. As the right end of i is more left than the right end of j, the left end of k must be more left than the right end of j, so clearly each probe in P_j is in $P_i \cup P_k$, which means $P_j \subseteq P_i \cup P_k$. □

In the proof we used that the clones are of equal length. From Lemma 2.1.1 follows that if $i' \leq i < j < k \leq k'$ and i does not overlap k, then i' does not overlap k'.

If i' and k' are such that there exist $i, j, k \in \mathcal{C}$ with $P_j \setminus (P_i \cup P_k) \neq \emptyset$ and $i' \leq i < j < k \leq k'$, then we say that the pair (i', k') is an **excluded pair**. In all other cases we say that the pair is **permitted**.

Lemma 2.1.2. *Let $i < j < k$ be 3 clones. If (j, k) is an excluded pair, then (i, k) is also excluded. If (i, k) is a permitted pair, then (j, k) is also permitted.*

We will show how to construct a feasible layout, where every permitted pair overlaps and no excluded pair does, further, it occupies a minimal span of the real line. This is the **greedy clone layout**.

The algorithm of greedy clone layout

First, choose a starting point, and place clone 1 here with its left end. Then for each $k \leq n$ find the smallest $i < k$ for which (i, k) is permitted. It is clear that such an i always exists, because $(k-1, k)$ is permitted. Now, place clone k in such a way that it overlaps with clone i, but it does not with clone $i-1$, and of course, its left end is at the right of the left end of clone $k-1$.

Gusfield [6] proved the following theorems:

Theorem 2.1.1. *At the end of the above algorithm, two clones overlap if and only if they form a permitted pair.*

To show that the greedy clone layout can be made feasible, we have to place the probes on the real line. The method of the placement of the probes is the following: Let i be the smallest index for which $p \in P_i$, and let j be the smallest index for

which $i < j$ and (i, j) is excluded or $p \notin P_j$. Then place a copy of p inside the clone i in such a way that all clones in $\{i, \ldots, j-1\}$ contain p, but j does not. If there is a $k \geq j$ for which p is in P_k, then restart the method with $i = k$.

Theorem 2.1.2. *It is possible to correct the above algorithm so that the result occupies less span than all other feasible layouts, and the above method makes this layout feasible.*

Chapter 3

Betweenness problem

In genome projects, it is important to order the mapping points if some betweenness constraints are given, such as one point is between two others. This problem is discussed by Gutin et al. [7]. The problem is the following: we have a set V of variables and a set $\mathcal{C}$ which contains the betweenness constraints. A betweenness constraint looks like v_i is between v_j and v_k (but is it not fixed if $v_j < v_k$ or $v_k < v_j$), and we sign it as $(v_i, \{v_j, v_k\})$. The aim is to show an ordering of V which satisfies as much betweenness constraints as much is possible. We will donate this ordering as an α bijection from V to the set $\{1, \ldots, |V|\}$, where α is called a **linear arrangement**.

Deciding whether all constraints can be satisfied is NP-hard, so this problem is NP-hard ([10]), and it is easy to see that the maximization problem is also NP-hard.

3.1 Parameterized problems

We call an L subset of $\Sigma^* \times \mathbb{N}$ a **parameterized problem** where Σ^* are the words formed from the finite alphabet Σ. We call the second component of an element of L a **parameter**.

If the question $(\alpha, k) \in L$ can be decided in $f(k) \cdot |x|^c$ time (where c is a real number), then we call L **fixed parameter tractable**.

The **kernelization** of an L betweenness problem is a polynomial algorithm which makes from an element $(x, k) \in L$ another element $(\chi, \kappa) \in L$, where the set of image of the algorithm is the **kernel**. This algorithm has to have the following properties:

first, it is an $L \to L$ injective function, it is contractive, so $\kappa \leq f(k)$, and $|\chi| \leq g(k)$ for some f and g functions. We say that $g(k)$ is the size of the kernel.

3.2 Parameterization of Betweenness problem

In the so-called *Betweenness problem* we ask if we could arrange linearly the elements by satisfying the most possible constraints. If we rather ask if there exists any arrangement that satisfies at least k constraints, we have parameterized the Betweenness problem with parameter k. Now we show that the Betweenness problem is fixed parameter tractable with this parameter. If the linear arrangement is a totally random permutation (with a probability of $\frac{1}{n!}$), then it is easy to see that it satisfies $\frac{|\mathcal{C}|}{3}$ constraints in expectation, hence if k is less than $\frac{|\mathcal{C}|}{3}$, the answer for parameterized betweenness problem is "yes". However, if $\mathcal{C}$ contains all the possible constraints, then no linear arrangement can satisfy more than $\frac{|\mathcal{C}|}{3}$ constraints.

We have seen that we can satisfy at least $\frac{|\mathcal{C}|}{3}$ constraints, so we can reparameterize the problem: we ask if we can satisfy at least $\frac{|\mathcal{C}|}{3} + k$ constraints. The name of this problem is **Betweenness Above Tight Lower Bound** (henceforth referred to as BATLB), and the question was opened by Benny Chor [4]. Gutin et al. [7] shows that BATLB is fixed parameter tractable, and actually it has a **kernel** of size $O(k^2)$, by using the **Strictly Above/Below Expectation Method** (henceforth referred to as SABEM).

3.3 The Strictly Above/Below Expectation Method

With SABEM we can prove whether a parameterized problem Π is above a tight lower bound. Firstly, SABEM reduces the problem, then it gives an X random variable which takes higher value than the parameter k with nonzero probability if and only if the answer for the reduced problem is "yes".

SABEM needs the following lemmas:

Lemma 3.3.1. *If the X random variable satisfies: $E(X) = 0$, $E(X^2) = \sigma^2 \neq 0$, $E(X^4) \leq c\sigma^4$, where c is a constant, then the probability $P(X > \frac{\sigma}{2\sqrt{c}}) > 0$.*

Lemma 3.3.2. *If f is a polynomial of the random variable $(X_1, \ldots, X_n) \in \{-1, 1\}^n$ with degree r and $X = f(X_1, \ldots, X_n)$, then $E(X^4) \leq 9^r E(X^2)^2$.*

3.4 FPT of BATLB

Gutin et al. [7] shows that BATLB has a quadratic kernel size. We denote the variables of the constraint C with $vars(C)$, and we say that a triplet of constraints A, B, C are complete if they have the same variables. If a complete triplet appears in $\mathcal{C}$, then we can remove this triplet with the variables which figures only in the triplet, because any linear arrangement satisfies exactly one constraint of the triplet. This is the **reduction rule**. Moreover, we can delete all the complete triplets from $\mathcal{C}$ one by one, and we call the final result (in which there are no complete triplets) **irreducible**.

Lemma 3.4.1. *If $(V, \mathcal{C}, k)$ is a BATLB-problem, and $(V', \mathcal{C}', k)$ is an irreducible BATLB-problem reduced from the original one, then the answer for $(V, \mathcal{C}, k)$ is "yes" if and only if it is "yes" for $(V', \mathcal{C}', k)$.*

Now let $\phi : V \to \{0, 1, 2, 3\}$ be a random function, and $\Lambda_i(\phi)$ is the set of the variables which are mapped into i by ϕ (for $i = 0, 1, 2, 3$). We will use the following notation: $\ell_i(\phi) = |\Lambda_i(\phi)|$. Let α be a bijection between V and $1, \ldots, |V|$, which randomly assigns the variables in $\Lambda_0(\phi)$ to $1, \ldots, \ell_i(\phi)$. Further, α assigns the variables of $\Lambda_i(\phi)$ to $\sum_{j=0}^{i-1} \ell_j(\phi) + 1, \ldots, \sum_{j=0}^{i} \ell_j(\phi)$. We call this special linear arrangement of the variables in V a **ϕ-compatible linear arrangement**.

Supposing that $\phi : V \to \{0, 1, 2, 3\}$ is a fixed function, α is a ϕ-compatible linear arrangement, we can define the following function for each $C \in \mathcal{C}$: $v_C(\alpha) = 1$ if the constraint is satisfied and 0 otherwise. Now, let $w(C, \phi) = E(v(\alpha)) - \frac{1}{3}$ and $w(\mathcal{C}, \phi) = \sum_{C \in \mathcal{C}} w(C, \phi)$.

Lemma 3.4.2. *[7] If $w(\mathcal{C}, \phi) \geq k$, then the answer for the BATLB-problem $(V, \mathcal{C}, k)$ is "yes".*

Proof. $k \leq w(\mathcal{C},\phi) = \sum_{C\in\mathcal{C}} w(C,\phi) = \sum_{C\in\mathcal{C}} E(v_C(\alpha)) - \frac{1}{3} = -\frac{|\mathcal{C}|}{3} + \sum_{C\in\mathcal{C}} E(v_C(\alpha))$. After reordering, we get $k + \frac{|\mathcal{C}|}{3} \leq \sum_{C\in\mathcal{C}} E(v_C(\alpha))$. Using the linearity of expectation, it is equivalent to $k+\frac{|\mathcal{C}|}{3} \leq E\left(\sum_{C\in\mathcal{C}} v_C(\alpha)\right)$, which means that in expectation at least $k + \frac{|\mathcal{C}|}{3}$ constraints are satisfied, so the answer is "yes".

Gutin et al. [7] also proofs the following three lemmas:

Lemma 3.4.3. *The expectation of $w(\mathcal{C},\phi)$ is zero.*

Lemma 3.4.4. *$w(\mathcal{C},\phi)$ is a polynomial with degree 6 and it satisfies the terms of Lemma 3.3.2.*

Lemma 3.4.5. *If the BATLB problem $(V,\mathcal{C},k)$ is irreducible, then $E(w^2(\mathcal{C},\phi)) \geq \frac{11}{768}|\mathcal{C}|$.*

Finally, we get the following result:

Theorem 3.4.1. *The size of the BATLB's kernel is $O(k^2)$.*

Proof. Let $(V,\mathcal{C})$ be a BATLB problem. First, we reduce it to an irreducible problem $(V',\mathcal{C}')$ in $O(m^3)$ steps (Lemma 3.4.1), where the answer for the original problem is "yes" if and only if it is "yes" for the reduced problem. The random variable $w(\mathcal{C}',\phi)$, which we defined above, is a polynomial with degree 6 (Lemma 3.4.4), so using Lemma 3.3.2 we get that $E\left(w^4(\mathcal{C}',\phi)\right) \leq 9^6 E\left(w^2(\mathcal{C}',\phi)\right)^2$. Further, using Lemma 3.3.1 and Lemma 3.4.5, we receive $P\left(w(\mathcal{C}',\phi) > \frac{1}{2\cdot 9^3}\sqrt{\frac{11}{768}|\mathcal{C}'|}\right) > 0$. Lemma 3.4.2 shows that if $\frac{1}{2\cdot 9^3}\sqrt{\frac{11}{768}|\mathcal{C}'|} \geq k$, then the answer for the reduced problem is "yes". Moreover, it is also shown that $|\mathcal{C}'| = O(k^2)$. □

Chapter 4

Comparison between different sequences

4.1 Exact matching

Let P be a string called **pattern** and let W also be a string called **text**. The **Exact matching** problem is to find all occurrences of P in T. For example, let P be a nucleotide sequence ACA and let W be a part of human mitochondrial tRNS: $ATACCTACACA$. P occurs twice in T, at positions 7 and 9. Of course, the occurrences of P might overlap.

Let $P(i, j)$ be a substring of P. If $i = 1$, then it is a **prefix** of P, and if $j = n$, where n is the length of P, then it is a **suffix** of P.

Gusfield [6] cites certain methods for solve the Exact matching problem.

4.1.1 The Naive Algorithm

The naive method aligns the left end of P to the left end of W, and it compares each pair of characters at the same position. One turn runs until the method finds a difference or it reaches the end of P, in which case an occurrence of P is found. In each case, P is shifted one place to the right and a new turn is started. The process comes to an end if P is longer than the remaining substring of W.

In the worst case, the number of comparisons is $n(m - n + 1)$, where $|P| = n$

and $|W| = m$. This running time is tight, for example if $W = AAAAAAA$ and $P = AAA$, then the method makes 12 comparisons.

4.1.2 The Boyer–Moore Algorithm

The basic principles of the Boyer–Moore algorithm are the same as in the naive algorithm, but it is updated by three new ideas, namely, the *right-to-left scan*, the *bad character shift rule* and the *good suffix shift rule*. This method runs in $O(m+n)$ steps, it is linear in the worst case.

The right-to-left scan

The Boyer–Moore algorithm starts the comparison at the right end of P, and the same happens as in the case of the naive method: we move leftwards until either a mismatch is found or the left end of P is reached, then P is shifted one place right. If the naive method is extended with this new rule, it is clear that the worst-case time remains $n(m - n + 1)$.

The bad character rule

Let $R(x)$ be the position of the right-most occurrence of x in P. Let $R(x)$ be zero if x does not appear in P. The $R(x)$ values could be collected in $O(n)$ time.

When the right-to-left scan finds a mismatch in the i. position of P and the k. position of W, the bad character rule comes into use. It shifts P right by $\max[1, i - R(W(k))]$ places. The advantage of this rule is that P is shifted more than one character if it is possible. After shifting, the method returns to the right end of P and restarts the comparison.

Clearly, the bad character rule is not too effective for small alphabets, e.g. nucleotides, but Boyer and Moore introduce a new rule for this purpose.

The good suffix rule

Assume that a substring w of W matches to a suffix $P(i, n)$ of P and at the next comparison (using the right-to-left scan) a mismatch is found. Let this substring be

w'. Now, search the right-most occurrence of w' in the prefix $P(1, n-i)$, and if it exists, shift P right so that w' in P is below substring w' in W. If w' does not exist, shift P so that the left end of P is placed at the start of w. In each case, restart the right-to-left scan. It is easy to see that using the good suffix rule no occurrence of P in W is missed.

Gusfield [6] shows that the worst-case runtime of the Boyer–Moore algorithm is $O(m)$.

The original Boyer–Moore algorithm can be found in [3], and uses another version of good suffix rule. The rule that we introduced is taken from [6].

4.1.3 The Knuth–Morris–Pratt Algorithm

The Knuth–Morris–Pratt algorithm is another improvement of the naive exact matching. We keep the left-to-right scan method.

The basic idea of this algorithm is the following: if only the location of the first mismatch is known, then P can be shifted by several places without knowing anything about W. For example, if $P = GACTAGCAGT$ and the first mismatch is at the position 7 (so at C), then P can be shifted at least 5 places right, because the distance between the first and the second G is 5.

To formalize the algorithm, we have to define sp_i $(i = 1, \ldots, n)$ to be the length of the longest proper suffix of $P[1..i]$ which matches a prefix of P *but* $P(i+1) \neq P(sp_i + 1)$.

If the first mismatch appears at the $i+1$. position of P and at the k. position of W, then shift P to the right with $i - sp_i$ positions. After the shift, the prefix $P(1, sp_i)$ aligns with the substring $W(k - sp_i, k-1)$. If an occurrence of P is found, then shift P to the right with $n - sp_n$ places. This rule provides a matching of the prefix $P(1, sp_i)$ with a substring of W. In the next step, $W(k)$ and $P(sp_i + 1)$ are compared.

Using the KMP shift rule is beneficial, because P is often shifted by more than one character, and after a shift, the left sp_i characters match with the aligned characters of W.

Gusfield [6] proved the following:

Theorem 4.1.1. *If the first i characters of P match the opposing characters of W but the $(i+1)$. character mismatches $W(k)$, then P can be shifted with $i - sp_i$ positions right, and no occurrence of P is passed.*

Theorem 4.1.1 states that using the KMP shift rule no occurrences of P in W can be missed, so it proves that the Knuth–Morris–Pratt-algorithm is correct. Gusfield also proves that the worst-case runtime is $2m$.

4.2 Inexact matching

4.2.1 Edit distances of two strings

Comparing two strings, often we are curious about the distance between them. There are a lot of ways to define the distance between strings. Now, we will review the **edit distance**, which concentrates on the transformation of one string into the other [6]. The transformation uses four edit operations: I as insertion, D as deletion, R as replacement and M as matching. There are given two input strings S_1 and S_2, the task is two show a transformation of S_1 into S_2 using these four operations.

Let $next_1$ and $next_2$ be two pointers to some characters in S_1 and S_2, respectively, and let their value be 1. Now we define what the above operations do. I inserts $S_2(next_2)$ after $S_1(next_1)$, then sets $next_1$ to $next_1 + 1$ and $next_2$ to $next_2 + 1$. D deletes $S_1(next_1)$ and raises $next_1$ with 1. R replaces $S_1(next_1)$ with $S_2(next_2)$, if $S_1(next_1) \neq S_2(next_2)$, else M matches them. The last two operations increase both pointers with 1.

We say that the **edit distance** of S_1 and S_2 is the minimum number of edit operations except M needed to transform S_1 into S_2. It is referred also as **Levenshtein distance**, since Levenshtein [9] discussed it first. An **optimal transcript** is an edit transcript constructed from the operations I, D, R and M using minimum numbers of operation.

Computing the edit distance

We will give a dynamic programming algorithm for compute the edit distance. Let S_1 and S_2 be two fixed strings. Let $D(i,j)$ be the edit distance of the prefixes $S_1(1,i)$ and $S_2(1,j)$. We will show a recursive relationship between the $D(i,j)$ values. Let $t(i,j)$ be 0 in case of $S_1(i) = S_2(i)$, otherwise 1.

The base conditions are $D(i,0) = i$ and $D(0,j) = j$, because if S_2 is empty, then each character of S_1 has to be deleted, and if S_1 is empty, then each character of S_2 has to be inserted into S_1.

Theorem 4.2.1.
$D(i,j) = \min\{D(i-1,j)+1, D(i,j-1)+1, D(i-1,j-1)+t(i,j)\}$

Proof. Let $D^*(i,j) = \min\{D(i-1,j)+1, D(i,j-1)+1, D(i-1,j-1)+t(i,j)\}$.

First, we prove $D(i,j) \geq D^*(i,j)$. Suppose indirectly that there is an optimal transcript of $S_1(1,i)$ to $S_2(1,j)$ the length l of which is less than $D^*(i,j)$. Let separate four cases according to the last operation of this transcript.

If this is deletion, then omitting this operation we get an $l-1$ long edit transcript of $S_1(1,i-1)$ to $S_2(1,j)$, which is less than $D(i-1,j)$, although we know that the length of a transcript of $S_1(1,i-1)$ to $S_2(1,j)$ is at least $D(i-1,j)$.

If the last operation is insertion, then omitting this operation we get an $l-1$ long edit transcript of $S_1(1,i)$ to $S_2(1,j-1)$, which is less than $D(i,j-1)$.

If the last operation is replacement or matching, then omitting this operation we get an $l-t(i,j)$ long edit transcript of $S_1(1,i-1)$ to $S_2(1,j-1)$, which is less than $D(i-1,j-1)$. In each case we ran into a contradiction, which means that $D(i,j) \geq D^*(i,j)$.

Now we have to prove $D(i,j) \leq D^*(i,j)$.

We will show three possible transformations of $S_1(1,i)$ to $S_2(1,j)$ using the quantities in $D^*(i,j)$.

First, transform $S_1(1,i)$ into $S_2(1,j-1)$ in $D(i,j-1)$ operations, and then insert $S_2(j)$. The length of this transcript is $D(i,j-1)+1$.

Second, transform $S_1(1,i-1)$ into $S_2(1,j)$ in $D(i-1,j)$ operations, and then delete $S_1(i)$. The length of this transcript is $D(i-1,j)+1$.

	-	A	A	C	C	T	G
-	0	$\leftarrow 1$	$\leftarrow 2$	$\leftarrow 3$	$\leftarrow 4$	$\leftarrow 5$	$\leftarrow 6$
A	$\uparrow 1$	$\nwarrow 0$	$\leftarrow\nwarrow 1$	$\leftarrow 2$	$\leftarrow 3$	$\leftarrow 4$	$\leftarrow 5$
C	$\uparrow 2$	$\uparrow 1$	$\nwarrow 1$	$\nwarrow 1$	$\leftarrow\nwarrow 2$	$\leftarrow 3$	$\leftarrow 4$
A	$\uparrow 3$	$\nwarrow\uparrow 2$	$\nwarrow 1$	$\leftarrow\nwarrow\uparrow 2$	$\nwarrow 2$	$\leftarrow\nwarrow 3$	$\leftarrow\nwarrow 4$
T	$\uparrow 4$	$\uparrow 3$	$\uparrow 2$	$\nwarrow 2$	$\leftarrow\nwarrow\uparrow 3$	$\nwarrow 2$	$\leftarrow 3$
T	$\uparrow 5$	$\uparrow 4$	$\uparrow 3$	$\nwarrow\uparrow 3$	$\nwarrow 3$	$\nwarrow\uparrow 3$	$\nwarrow 3$
G	$\uparrow 6$	$\uparrow 5$	$\uparrow 4$	$\nwarrow\uparrow 4$	$\nwarrow\uparrow 4$	$\nwarrow\uparrow 4$	$\nwarrow 3$

Table 4.1: Edit distances

Third, transform $S_1(1, i-1)$ into $S_2(1, j-1)$ in $D(i-1, j-1)$ operations, and then if $S_1(i) = S_2(j)$ then match, else replace $S_1(i)$ to $S_2(j)$. The length of this transcript is $D(i-1, j-1) + t(i, j)$.

As we have this transformations, there exists an edit transcript with $D^*(i, j)$ operations, hence we got that $D(i, j) \leq D^*(i, j)$, and using our first result $D(i, j) = D^*(i, j)$. □

Using Theorem 4.2.1 we can compute the edit distance of S_1 and S_2. We will construct a tabular of edit distances. In the intersection of row i and column j we write $D(i, j)$. An illustration for the edit distance tabular for the strings $AACCTG$ and $ACATTG$ from [6] can be seen in Table 4.1. We place a pointer into each cell by the following rule. Set a pointer from cell (i, j) to cell $(i-1, j)$ if $D(i, j) = D(i-1, j)+1$, set a pointer from cell (i, j) to cell $(i, j-1)$ if $D(i, j) = D(i, j-1)+1$ and respectively, set a pointer from cell (i, j) to cell $(i-1, j-1)$ if $D(i, j) = D(i-1, j-1) + t(i, j)$. In each cell we find the value $D(i, j)$, the edit distance of $S_1(1, i)$ and $S_2(1, j)$.

Having the tabular, all optimal transcripts can be found by following the pointers from cell (n, m) backwards to cell $(0, 0)$.

Weighted edit distances

A generalization of edit distance is associating a **weight** or **cost** to each edit operation. Let the insertion and the deletion step have weight d, a replacement has

weight r and a matching has weight m. Usually m is much smaller than the other weights. The task is to find an edit transcript that transforms S_1 into S_2 with the minimum total operation weight.

The original edit distance is a special case of the weighted one with $d = r = 1$ and $m = 0$.

Clearly, if $r > 2d$ then an optimal edit transcript does not contain any replacements, because a replacement can be substituted for a deletion and an insertion.

We have a recursive formula for computing the operation-weighted edit distance. Let $D(i, j)$ denote the weighted edit distance of $S_1(1, i)$ and $S_2(1, j)$ and let $t(i, j)$ be m if $S_1(i) = S_2(j)$ else let it be r. It is easy to see that $D(i, 0) = id$ and $D(0, j) = jd$. The recursion is $D(i, j) = \min\{D(i, j-1) + d, D(i-1, j) + d, D(i-1, j-1) + t(i, j)\}$. This statement can be proved similarly than 4.2.1.

In genetics, it often occurs that the weight depends on exactly which character is removed and which is added. For example, transforming A into T is more costly than into G. This distance is **alphabet-weighted edit distance**. Of course, operation-weighted edit distance is a special alphabet-weighted edit distance.

4.2.2 Representing DNA sequences with matrices

Randić [12] makes a representation of DNA sequences using 4×4 matrices constructed in a special way, and he states that we can compare two sequences by comparing several invariants of the matrices.

First, we make an $n \times n$ matrix (SD) from the serial distances of the sequence (see Table 4.2 for the sequence Shine–Dalgarno [8]). That is, the i. row refers to the i. position of the sequence and the j. column refers to the j. element (A, G, C or T) of the sequence. $a_{ij} = k$ if after the i. position the j. element of the sequence is the k. of its type (so the same nucleotide base), when $j > i$. When $j < i$, let a_{ij} be a_{ji}. In case of $j = i$, $a_{ij} = 0$. In the table only the part above the main diagonal is represented.

Then we rearrange the $n \times n$ matrix using the next method: the rows are classified by the appropriate base (so the first few columns are A, the next few are C, then T

	G1	A1	T1	T2	C1	C2	T3	A2	G2	G3	A4	G4	G5	T7	T8	T9
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	1	1	2	1	2	3	2	1	2	3	3	4	4	5	6
2		0	1	2	1	2	3	1	1	2	2	3	4	4	5	6
3			0	1	1	2	2	1	1	2	2	3	4	3	4	5
4				0	1	2	1	1	1	2	2	3	4	2	3	4
5					0	1	1	1	1	2	2	3	4	2	3	4
6						0	1	1	1	2	2	3	4	2	3	4
7							0	1	1	2	2	3	4	1	2	3
8								0	1	2	1	3	4	1	2	3
9									0	1	1	2	3	1	2	3
10										0	1	1	2	1	2	3
11											0	1	2	1	2	3
12												0	1	1	2	3
13													0	1	2	3
14														0	1	2
15															0	1
16																0

Table 4.2: The 16×16 matrix created from the serial distances (SD)

	A1	A2	A3	G1	G2	G3	G4	G5	C1	C2	T1	T2	T3	T4	T5	T6
	2	8	11	1	9	10	12	13	5	6	3	4	7	14	15	16
2	0	1	2	1	1	2	3	4	1	2	1	2	3	4	5	6
8		0	1	2	1	2	3	4	1	1	1	1	1	1	2	3
11			0	3	1	1	1	2	2	2	2	2	2	1	2	3
1				0	1	2	3	4	1	2	1	2	3	4	5	6
9					0	1	2	3	1	1	1	1	1	1	2	3
10						0	1	2	2	2	2	2	2	1	2	3
12							0	1	3	3	3	3	3	1	2	3
13								0	4	4	4	4	4	1	2	3
5									0	1	1	1	1	2	3	4
6										0	2	2	1	2	3	4
3											0	1	2	3	4	5
4												0	1	2	3	4
7													0	1	2	3
14														0	1	2
15															0	1
16																0

Table 4.3: The reordered serial distance matrix, RSD

and G respectively). Let i^* be $\min(i, j)$ and j^* be $\max(i, j)$. The ij. element of the reordered matrix (RSD) is the i^*j^*. element of the matrix SD (See Table 4.3).

In the next step we create the S/S matrix. The ij. element of the S/S matrix is RSD_ij/s, where RSD_ij is the ij. element of the reordered serial distance matrix and $s = |i - j|$ (See Table 4.4). From this matrix, we create the MN submatrices (where $M, N \in A, C, T, G$) (see Table . The ij. element of the AA submatrix is $s/(j - i)$, where s is the serial distance of the i. and the j. A. Respectively, we can create all the 16 submatrices. The size of the MN submatrix is $m \times n$ if in the sequence there are m M and n N bases. It is clear that $MN = NM^T$.

	A1	A2	A3	G1	G2	G3	G4	G5	C1	C2	T1	T2	T3	T4	T5	T6
	2	8	11	1	9	10	12	13	5	6	3	4	7	14	15	16
2	0/0	1/6	2/9	1/1	1/7	2/8	3/10	4/11	1/3	2/4	1/1	2/2	3/5	4/12	5/13	6/14
8		0/0	1/3	2/7	1/1	2/2	3/4	4/5	1/3	1/2	1/5	1/4	1/1	1/6	2/7	3/8
11			0/0	3/10	1/2	1/1	1/1	2/2	2/6	2/5	2/8	2/7	2/4	1/3	2/4	3/5
1				0/0	1/8	2/9	3/11	4/12	1/4	2/5	1/2	2/3	3/6	4/13	5/14	6/15
9					0/0	1/1	2/3	3/4	1/4	1/3	1/6	1/5	1/2	1/5	2/6	3/7
10						0/0	1/2	2/3	2/5	2/4	2/7	2/6	2/3	1/4	2/5	3/6
12							0/0	1/1	3/7	3/6	3/9	3/8	3/5	1/2	2/3	3/4
13								0/0	4/8	4/7	4/10	4/9	4/6	1/1	2/2	3/3
5									0/0	1/1	1/2	1/1	1/2	2/9	3/10	4/11
6										0/0	2/3	2/2	1/1	2/8	3/9	4/10
3											0/0	1/1	2/4	3/11	4/12	5/13
4												0/0	1/3	2/10	3/11	4/12
7													0/0	1/7	2/8	3/9
14														0/0	1/1	2/2
15															0/0	1/1
16																0/0

Table 4.4: The S/S matrix

Now we define a matrix invariant: let W be the average of all elements in the matrix. Compute W for all submatrices, and create the condensed matrix from this 16 data (See Table 4.5). The condensed matrix is symmetric, so it is enough to store only the part above the main diagonal.

	A	G	C	T
A	0.1604	0.6461	0.4	0.4579
G		0.4429	0.0401	0.0163
C			0.5	0.0454
T				0.4087

Table 4.5: The condensed matrix

Randić expected that different sequences lead to different condensed matrices, but this is not proved. The other problem of this reduced data storing is that it is not clear if any biological information can be restored of the condensed matrix, however, this may be possible.

Summary

As a conclusion, it is clear that the communication between mathematicians and biologists should be enhanced. Of course, this requires patience and compromises from both parts. Nowadays, however, the tendencies are prospering.

The interpretation of results of different mathematical methods is very interesting. It is also possible that a result cannot be biologically interpreted. We should make an effort to give a clear and true biological interpretation of mathematical methods.

On the other hand, we can get different answers for different questions. For example, the distance of two sequences is not a well-defined function.

All things considered, the interdisciplinary field biomathematics is very interesting and full of promise.

Bibliography

[1] Alizadeh, F., Karp, R., Weisser, D., Zweig, G., *Physical mapping of chromosomes using unique probes*, Journal of Computational Biology, 51, 1990, pp. 431-453

[2] Booth, K., Lueker, G., *Testing for the consecutive ones property, interval graphs and graph planarity testing using pq-tree algorithms*, Journal of Computer and System Sciences, 13, 1976, pp. 333-379

[3] Boyer, R.S., Moore, J.S., *A fast string searching algorithm*, Communications of the ACM, 1977, pp. 762-772

[4] Chor, B., Sudan, M., *A geometric approach to betweenness*, SIAM Journal of Discrete Mathematics, 11, 1998, pp. 511-523

[5] Green, E., Green, P., *Sequence-tagged site (STS) content mapping of human chromosomes: theoretical considerations and early experiences*, PCR Methods and Applications, 1991, pp. 77-90

[6] Gusfield, D., *Algorithms on strings, trees and sequences*, Cambridge University Press, 1999, pp. 5-29, 215-225, 395-412

[7] Gutin, G., et al., *Betweenness parameterized above tight lower bound*, Journal of Computer and System Sciences, 2010

[8] Láng, F., et al., *Növényélettan, A növényi anyagcsere II.*, **in** Hungarian, 2007, p. 695

[9] Levenshtein, V. I., *Binary codes capable of correcting insertions and reversals*, Soviet Physics – Doklady, 10, 1996, pp. 707-710

[10] Opatrny, J., *Total ordering problem*, SIAM Journal of Computation, 8, 1979, pp. 111-114

[11] Orosz, L., *Klasszikus és molekuláris genetika*, **in** Hungarian, Akadémiai Kiadó, 1980

[12] Randić, M., *On characterization of DNA primary sequences by a condensed matrix*, Chemical Physics Letters, 317, 2000, pp. 29-34

[13] Weinberg, R., *Finding the anti-oncogene*, Scientific American, 1998, pp. 44-51

[14] Williams, N., *Europe opens institute to deal with gene data deluge*, Science, 269, 1995, p. 630