Università di Pisa
Dipartimento di Informatica

Technical Report: TR-08-21

BpMatch: an efficient algorithm for segmenting sequences, calculating genomic distance and counting repeats

Claudio Felicioli Roberto Marangoni

Giugno 2007

address: Largo B. Pontecorvo 3, 56127 Pisa, Italy. tel: +39 050 2212700 fax: +39 050 2212726

BpMatch: an efficient algorithm for segmenting sequences, calculating genomic distance and counting repeats

Claudio Felicioli Roberto Marangoni

Giugno 2007

Abstract

There are several important reasons (biological, evolutionary, clinical, etc.) to give a segment-based description of genomic sequences, and, in particular, to detect repeated segments, written both direct and complemented inverted. In some applications, in particular in medical genomics, it is also necessary to count the number of occurrences of a segment. Moreover, by detecting common segments shared by two different sequences it is possible to define a sort of genomic distance between them. Here we propose BpMatch: an algorithm that, working on a suitably modified suffix-tree data structure, allows us to achieve all these three goals (identify repeated segments, including the complemented inverted copies of them, count repeats number and calculate genomic distance) in a fast and efficient way. BpMatch is able to identify exact copies (and complemented inverted copies) of a segment. The operator should define *a priori* the minimum length l of a string, in order to be considered a segment, and the minimum number of occurrences $minRep$, so that only segments having a number of occurrences greater than $minRep$ are considered to be significant. BpMatch is very efficient; we determined the complexity in time to calculate the self-covering of a string S, giving l, the alphabet dimension d and $n = |S|$. On the worst case, assuming the alphabet dimension is a constant, the time required to calculate the coverage is $\mathrm{O}(l^2 n)$. On the average, using $l \geq 2\log_d(n)$, the time required to calculate the coverage is only $\mathrm{O}(n)$. It is important to note that this estimation includes the time required to complete all of the three different tasks: to identify copied segments, to localize them, to count the number of occurrences and to evaluate the sequence coverage.

1 Introduction

During the pre-genomic era, the problem of comparing two biological sequences, in order to infer possible molecular evolution relationships, has been attacked following the aligment approach. Local and global aligment algorithms have

been proposed (to cite only the classic ones: Needelman and Wunsch; Pearson and Lipman), and continuosly improved in their heuristic implementations, in order to find the optimal aligment in the shortest possible time. Most of these algorithms are based on the same underlying biological hypothesis: the main molecular evolution process is the so-called "point mutation", the replacement of a single character with another. Finding an optimal character-by-character aligment is, therefore, the way to compare DNA and protein sequences. The only exception to this rule, is given by the possibility to insert gaps: by inserting several consecutive gaps the alignment process can take into account a different type of molecular evolution phenomenon: the insertion/deletion of a segment (subsequence).

In the current post-genomic era, the biological scenario is completely changed, in particular for DNA sequences analysis. At genomic level, in fact, point mutation does not represent the main, nor the most frequent, molecular evolution process: segment translocation, segment complementation and inversion (when a segment is written in the reversed order, and in the complementary bases), segment insertion and deletion, segment duplication, are more frequent and quantitatively relevant (see Liu 1998).

To better describe evolutionary relationships between genomic sequences the focus should be moved from characters to segments. In this perspective, by analogy with Image Analysis or signal theory in general, different previously published papers are addressing the concept of "segmenting sequences" (Kedzierska and Husmeier 2006; Keith 2006; Azad *et al.* 2002; Li 2001; Andre *et al.* 2001). Their approaches differ very widely with respect to what should be considered a segment.

In the present paper, to define what is a segment, we start from the consideration of the existence of certain similarities among particular sequences playing important biological functions: operators, promoters, TFBSs (transcritpion factor binding sites), transposons, etc. In most of these cases, in fact, it is possible to detect the presence of such sequences thanks to some specifity in their structure: for example, many TFBSs, and most of the transposon edge regions, are frequently made of short series of repeated segments, followed by (or close to) some repeats of inverted complemented copies of the same segments. This particular sequence organization is due to the necessity of the DNA to assume special 3D conformations such as: hairpins, loops, etc. (see Lewin 2003)

For all these situations, we define a segment as any subsequences that appear to be repeated either direct or inverted complemented. Our first goal, therefore, is to identify and localize all the occurrences of repeated segments, and complemented inverted repeated segments.

Moreover, clinic researches have shown that some genetic diseases are generated by the increase or decrease of the number of repeats of defined segments. They can span from a series of triplets, entire genes or even long genomic regions (Redon *et al.* 2006; Komura *et al.* 2006; Sharp *et al.* 2005; Stranger *et al.* 2007). The importance of counting the number of repeats is so high, that research lines aimed at studying Copy Number Variation (CNV) have been hosted by all the most important international molecular biology organizations, in order

to screen genomic variants in human species and make them available though suitable databases (see, for example, the Database of Genomic variants, available at: http://project.tcag.ca/variation and described in Zhang *et al.* 2006). In other words, medical genomics needs not only to identify repeated segments, but also to count the number of repetitions.
By working with segments, it is also possible to define a quantitative measurement of the genomic evolutionary distance between two sequences: generally speaking, it is given by the percentage of covering one sequence using as many segments as possible taken from the other. Following this approach Varré *et al.* (1999) proposed Transformation Distance, a scripting distance that, given two sequences S and T and a set of elementary operations made on segments (Copy, ReverseCopy, Insert), each of which has an associated cost. TD(S, T) is then defined as the cost of the script of minimum cost, by which the string T is generated using, where possible, segments derived from the string S. TD is an uncommon type of evolutive distance, in particular, different from commonly used aligment distances, which verify simmetry condition, TD is asymmetric. This feature is exploitable for some biological applications, for example in the attempt to reconstruct the history of gene duplication events, where the relationship template-copy is intrinsically asymmetric (see Pisanti *et al.* 2003). Unfortunately, the TD algorithm complexity is too high to use TD in practice. In fact, the algorithm presented to calculate TD, which ignores possible segment copy overlaps, has a time complexity of $O(n^6)$ on the worst case and a high (although not analyzed) spatial complexity which makes it generally inapplicable on sequences longer than 100Kbp. Another algorithm (*compact script graph*) is mentioned in (Varré *et al.* 1999) showing a graph in which the TD calculation seems to be quadratic in time without regard to sequences length. A precise analysis of this second version is not indicated, and no spatial complexity has been calculated. The high computational cost of the algorithms to calculate TD has prevented its widespread use, even though they represent a valid attempt to give a correct reconstruction of the molecular events that cause divergence between genomic sequences.

We hereby propose a new algorithm (called BpMatch) that, given two sequences S and T, finds a covering of T, using only segments and reversed complemented segments of S, eventually overlapped. BpMatch works with the constraint that the segments must be at least l characters long and must be repeated in S at least $minRep$ times, such a covering is the maximal in coverage and the number of segment utilized is the minimal.
The BpMatch algorithm, thanks to its very low time and space complexity, is our first achievement. The output of BpMatch is represented by a list of identified repeated segments and their respective position, and an overall evaluation of the achievable maximum coverage. In practice, BpMatch performs, in a very fast time, a segmentation of the two sequences and calculates a mutation distance measure.
This approach can be used also to have a repetitions analysis on a single sequence, using a *selfcovering* approach, by which we can count the number of

repeated segments included in a sequence. BpMatch is able to indicate which segments have been extracted from the sequence, in which position they are, and how many copies have been counted.
In the discussion section of the present paper, the performances of BpMatch will be compared to those of TD, since the goal, strategy and output of these two algorithms are very similar to each other.

2 Approach

2.1 The covering as a mutation distance measure

The BpMatch algorithm solves a problem similar to that of TD: the maximum covering of a *target* sequence T using the segments and the reversed complemented segments of a *source* sequence S can be interpreted as an estimation of the part of T that can be derived from S through sequence genomic mutation. The mutation distance derived from the covering is then proportional to the number of segments and to the ratio between covered part and non covered part lengths.

2.2 Repetitions analysis

Our approach in repetitions analysis consists in the application of BpMatch algorithm using for target the same sequence used for source.

Analysing the coverage variations obtained varying l (minimum segment length) and $minRep$ (minimum segment repetitions) parameters, the repetition zones can be characterized.

3 Methods

3.1 BpMatch algorithm

The calculation of T's maximum coverage, using only the minimized number of segments and complemented reversed segments of S, with minimum length of l and repeated in S at least $minRep$ times, possibly overlapped, requires the use of two data structures: one capable to recognize the segments of S and the other capable to recognize the segments of S complemented reversed; these two structures should also be able to count the number of repeats of the recognized segments. The suffix-tree (Weiner 1973; McCreight 1976) is a data structure suitable for these purposes. It is, in its usual form, already able to recognize segments: in order to make it also able to rapidly count them, it is enough to introduce an improvement: each inner node should keep note of the number of leaves of the sub-tree rooted in it. During the recognizing process of a segment, the number annotated in the reached node represents exactly the repetitions number of that segment (each occurrence of the segment has a different suffix,

which has that segment as prefix, and every different suffix ends in a different leaf.

We call G and G' the suffix trees calculated from S and from the reverse complement of S, annotated, as previously described, to be able to count the number of repetition of each recognized segment.

The algorithm processes the sequence T, it starts by considering the first character as a possible beginning for a segment and proceeds for iterations, processing ahead by one or more character steps for each iteration, until it catches up the end of T.

During each iteration two cases can take place:

case 1: the character immediately previous to that currently analyzed can not be covered by any segment

case 2: a segment ends exactly before the currently analyzed character.

In the first iteration we are in case 1.

c_0 is defined as the currently analyzed character and c_i as the character distant ahead i positions from c_0, $\overline{c}$ is complemented c.

3.1.1 Case 1

The hypothesis that the character c_{-1}, immediately before the currently analyzed one, can not be covered by any segment is verified, therefore c_0 can be covered only if it is the starting character of a covering segment.

Let us start by searching a direct segment of length at least l, starting from c_0, covering T as far as at least c_{l-1}. Such a segment exists only if we try to find a path in G' processing in order the complemented characters $\overline{c_{l-1}}$ $\overline{c_{l-2}}$ ... $\overline{c_0}$.

If the operation of recognition through G' is successful, then c_0 is the first character of a direct segment of length at least l, so let us start by searching such a direct segment, trying to find a path in G processing in order c_0 c_1 ... until, processing c_i (surely $i \geq l$, thanks to the previous G' utilization), it fails to find a path, therefore the direct S's segment c_0 c_1 ... c_{i-1} is found.

Otherwise, if the research of the reversed segment starting from c_{l-1} fails during the process of a certain c_i, this means that c_0 can not be covered by any direct segment of the minimum length l, because such a segment must contain the segment c_i c_{i+1} ... c_{l-1} which, since it is not an S's direct segment (since it is not recognized by G' its complemented reversion), can not be contained in any S's direct segment and so, from the case 1 hypothesis, c_0 can not be covered by any direct segment, but this means that c_1 falls back in case 1 hypothesis (at least for the direct segments) too, and, for the same reasons, it can not be covered also by direct segments. Therefore all the characters up to c_i included can not be covered by direct segments.

In order to find a complemented reversed segment of S, starting from c_0, the same procedure has to be used, but using G in place of G' and vice versa.

After that, three distinct situations can occur :

1. If only a segment is found, direct or complemented reverse, starting from c_0, then it will be added to the coverage and the character immediately after such a segment has to be processed by following the procedure described on case 2.

2. If both the direct and the complemented reverse segments are found, then the longer one of them will be added to the coverage and the character immediately after such a segment has to be processed by following the procedure described on case 2.

3. If neither the direct nor the complemented reverse segments are found, then all the characters that can not be covered by neither direct nor complemented reverse segments have to be excluded and the character immediately after these has to be processed by following the procedure described on case 1.

3.1.2 Case 2

The hypothesis that the character c_{-1}, immediately before the currently analyzed one, is the last character of a covering segment is verified, therefore all the previous characters up to c_{-l} (included) are surely covered and c_0 can be not only the first character of a new segment, but it can occur in a central position of a segment, of length at least l, partially overlapped to the segment ending at character c_{-1}.

In order to find the best direct segment covering c_0, if it exists, $c_0\ c_1\ ...\ c_i$ are processed by G until, during the c_i processing, it fails to find a path.

If $i \geq l$ then it is found: such a segment can also be just the final part of an overlapping segment, but it is obvious that there does not exist any overlapping segment that includes the characters from c_0 to c_i because $c_0\ c_1\ ...\ c_i$ is not recognized as a direct S's segment, so every segment having $c_0\ c_1\ ...\ c_i$ as one of its substrings can not be recognized either.

If instead $i < l$, then we have to make a search for a partial overlapping direct segment, covering the maximum possible amount of not yet covered characters and known to be unable to cover c_i (not being $c_0\ c_1\ ...\ c_i$ a direct segment).

Characters $\overline{c_0}\ \overline{c_{-1}}\ ...\ \overline{c_{-k}}$ have to be processed by G' while $k < l$ or until a fail in suffix tree's path finding occurs , so $c_{-1}\ c_{-2}\ ...\ c_{-k}$ is a reverse S's segment and it is the longest totally overlapped prefix (of length k) of the direct partially overlapping segment we are searching for.

The longest suffix $(c_0\ c_1\ ...\ c_{i-1})$ is of length i, then if $k + i < l$ the direct segment does not exists, else $c_{i-l}\ c_{i-l+1}\ ...\ c_{i-1}$ has to be processed by G.

If the sequence is recognized then a direct segment covering all the maximum suffix is found, else, if it fails in suffix tree's path finding at c_j, it means that $c_{i-l}\ c_{i-l+1}\ ...\ c_j$ is not a direct S's segment and therefore, since it is not able to be included in any direct segment of minimum length l, as c_{i-l+1} can not be the starting character of a direct segment arriving to c_{i-1} (it would be at most of length $l-1$), the length of the longest suffix need to be reduced from i to j.

If $k + j < l$ then the direct segment does not exist, else the same procedure has to be repeated replacing i with j.

On the worst and very improbable case, it is necessary to repeat this procedure for a maximum $l - 1$ times in order to find, if it exists, the searched direct and partially (in this extremized case, almost totally) overlapping segment.

In order to find a complemented reversed segment of S, starting from c_0, the same procedure has to be used, but using G in place of G' and vice versa.

After that, three distinct situations can occur :

1 - If only a segment is found, direct or complemented reverse, starting from c_0, then it will be added to the coverage and the character immediately after such a segment has to be processed by following the procedure described on case 2.

2 - If both the direct and the complemented reverse segments are found, then the longer one of them will be added to the coverage and the character immediately after such a segment has to be processed by following the procedure described on case 2.

3 - If neither the direct nor the complemented reverse segments are found, then c_0 can not be covered by neither direct nor complemented reverse segments and c_1 has to be processed by following the procedure described on case 1.

3.1.3 Using only segments repeated in S at least *minRep* times

For the sake of clearness the BpMatch algorithm was described in the simpler case in wich $minRep = 1$, that is the default value when covering to estimate a mutation distance measure (section 2.1).

The complete algorithm can be obtained trivially excluding from the path finding in the suffix tree all the edges that connect to a node with the annoted value less than *minRep*.

3.1.4 An example

Figure 1 graphically shows an execution of the maximum coverage calculation algorithm BpMatch. The alphabet used is A C G T, $\overline{\text{A}}$ = T and $\overline{\text{C}}$ = G, $l = 4$, $minRep = 1$, S="TCCATCCCCA", T="AGTCCTGAATCCCATCG", the arrows over the segments represent a segment recognition executed using the specified suffix tree, the cross on the arrows stands for the character the recognition of which has failed.

Figure 2 shows the two suffix trees used by the BpMatch algorithm.

3.2 Selfcovering algorithm

The application of BpMatch algorithm using as a target the same sequence S used as a source is called $selfcovering$. The selfcovering analysis approach consists of executing selfcovering varying l and *minRep* parameters until all the (l, *minRep*) pairs of interest are used. To obtain this the suffix trees are generated once, then BpMatch is executed for each (l, *minRep*).

4 Results and Discussion

4.1 Sequences segmentation: an application example

To give a segment-based description of a gene family it is enough to concatenate all the gene sequences into a single string, then to explore the selfcovering of it. For instance, the **wnt** gene family in human has been written as a single string. The main BpMatch output is an ASCII file, scrutured as a table, where, for each segment found, is declared which is the segment and where it appears on the sequence. To have a visual report of the content of the file, one can elaborate BpMatch's output to have a graphic display of the occurrences of a selected segment: an example is shown in Fig. 3, where it is possible to notice that direct and complemented inverted occurrences are shown using different colors.

4.2 Complexity analysis

Let $n = |S|$, and let d be the alphabet cardinality: the generation cost for both the two suffix trees from S and reverse complement S is $\mathrm{O}(n)$; the required space is $\mathrm{O}(n)$ as well. The cost for a match with a sequence long m is $\mathrm{O}(m \log(d))$, or even $\mathrm{O}(m)$, when the alphabet is supposed to be constant (Apostolico 1985; Stephen 1994; Grossi and Vitter 2005).

By adding, to the suffix tree nodes, the suitable annotations to make a fast occurrences counting for every recognized segment, the increase in the data structure space is linear with the suffix-tree dimension. Also the increase in its building time is linear, as a simple depth-first inspection allows us to calculate all the nodes annotations.

The calculation of T's maximum coverage requires to process $u = |T|$ character, switching from case 1 to case 2 and *vice versa*. Both the time required to examine a case and the number of character we can move forward depend on the case number.
On case 1:

- on failure : the time to match a segment of length m ($m \leq l$), moving forward $l - m + 1$ characters.
- on success : the time to match a segment of length l and then one other of length m ($m > l$), moving forward $m - 1$ characters.

On case 2:

- on failure : moving forward 1 character, on the worst case $l + 1$ segments of length l need to be matched.
- on success, finding a non overlapped segment : the time to match a segment of length m ($m > l$), moving forward $m - 1$ characters.
- on success, finding an overlapped segment : the situation on the worst case can be as in the failure case.

Thus, on the worst case, assuming the alphabet dimension is a constant, the time required to calculate the coverage is $O(l^2u)$. On the average, using $l = q\log_d(n)$ ($q \geq 2$, $l << u$, needed in order to obtain a statistically significant S-T comparison, as described in section 4.4), the time required to calculate the coverage is $O(u/q)$.

4.3 Distance Measure Symmetry

As it happens for TD, also the distance calculated by BpMatch is asymmetric. This is an unaivoidable consequence of the common approach segment-based, and, as recalled in the Introduction section, it is directly applicable in biological situations which are intimately asymmetric, as, for example, the relationship template seq./ copy seq. in a gene duplication process. BpMatch can therefore be used for all the same applications for which TD is used, with the advantage of a strong reduction of the complexity.

4.4 Minimum length l : meaning and quantification

In order to get a statistically significant S-T comparison, an $l > \log_d(n)$ ($n = |S|$) needs to be used, with d the alphabet cardinality.

The greater is l, the more reliable is the coverage. On the other hand, the more is the coverage reliability, the less is the sensibility, this because we do not consider all the segments of length less than l and, among these, also those who could have the role that we are seeking. The l value needs to be chosen while keeping in mind the reliability/sensibility ratio we want to obtain.

It is observed that, even if the case 2 sub loop of the BpMatch algorithm can cause a great increase of the computational complexity of the algorithm, such an increase is present only when the value of l is near to $\log_d(n)$.

Some tests, using either pseudo-random generated genomic sequences and biological genomic sequences as S and T, varying l over many different values, experimentally confirm that, to get good reliability of the coverage, the value of l must been bigger than the computationally critic zone $\log_d(n)$.

Figures 4 and 5 show two examples of these tests, using S and T of length 1000000: the $x - axis$ represent the different values of l (from 7 to 20 and from 20 to 50) and the height of the column above it represents the number of iterations of the resulting covering algorithm run. Each column is divided in three different gray levels, the dark gray area indicates the number of case1 iterations, the light gray area the number of case2 iterations and the black area the number of case2 sub-loop iterations. The figure on the right is the same, rescaled.

The number of sequences of the covering, for each l value, is intuitively represented by the light gray area; for small l all T is covered, for big l T is covered only if there is a segment copy relationship between S and T, the $case2$ sub-loop, that make the algorithm complexity quadratic, appear to be influent only for l near the critical value of $\log_d(n)$.

Using $l = 20 \simeq 2\log_4(1000000) = 2\log_d(n)$ the coverage results as not empty only comparing pieces of biological genomic sequences.

5 Conclusions

BpMatch has been proved to be an efficient algorithm that, with only one computation, can address three different problems of a high interest in modern genomics. We note that, through the selfcovering approach is also possible to use BpMatch to discriminate between coding (usually pesudo-random) and non-coding (usually contanining large numbers of repeats) regions in a genome. Even though BpMatch has not be designed as a gene-finder algorithm, preliminary observations show that it can be used with this purpose, in order to have a preliminar and fast classification of genomic tracts.

6 Aknowledgments

The Authors thank Luca Freschi for his contribution in the practical tests of BpMatch.
This work has been partly supported by Italian MIUR grant Italy-Israel FIRB "Pattern Discovery Algorithms in Discrete Structures, with application to Bioinformatics".

7 References

Andre, C., Vincens, P., Boisvieux, J.F., and Hazout, S. 2001. MOSAIC: segmenting multiple aligned DNA sequences, *Bioinformatics* 17, 196-197.

Apostolico, A. 1985. The myriad virtues of subwords trees, 85-95. *In* A. Apostolico and Z. Galil eds.,*Combinatorial Algorithms on Words. Nato Asi Series, Advanced Science Institutes Series, Series F, Computer and Systems Sciences, Vol 12*, Springer Verlag, New York.

Azad, R.K., Bernaola-Galvan, P., Ramaswamy, R., and Rao, J.S. 2002. Segmentation of genomic DNA through entropic divergence: Power laws and scaling, *Phys. Rev. E* 6505, 1909-1909.

Grossi, R., and Vitter, J.S. 2005. Compressed suffix arrays and suffix trees with applications to text indexing and string matching, *SIAM Journal on Computing* 35(2), 378-407.

Kedzierska, A., and Husmeier, D. 2006. A heuristic bayesian method for segmenting DNA sequence alignments and detecting evidence for recombination and gene conversion. *Statistical Application in Genetics and Molecular Biology* 5, 65-97.

Keith, J. M. 2006. Segmenting eukaryotic genomes with the generalized Gibbs sampler, *J. Comp. Biol.* 13, 1369-1383.

Komura, D., Shen. F., Ishikawa, S., *et al.* 2006. Genome-wide detection of human copy number variations using high-density DNA oligonucleotide arrays, *Genome Res.* 16, 1575-1584.

Lewin, B. 2003. *Genes (VIII ed.)*, Prentice Hall, New York.

Li, W. T. 2001. New stopping criteria for segmenting DNA sequences, *Phys. Rev. Lett.* 86, 5815-5818.

Liu, B. H. 1998. *Statistical Genomics*, CRC Press, Boca Raton, (FL, USA).

McCreight,E.M. 1976. A space-economical suffix-tree construction algorithm, *Journal of the ACM* 23(2), 262-272.

Pisanti, N., Marangoni, R., Ferragina, P., Frangioni, A., Savona, A., Pisanelli, C., and Luccio, F. 2003. PaTre: A Method for Paralogy Trees Construction, *J. Comp. Biol.* 10, 791-802.

Redon. R., Ishikawa, S., and Fitch, K.R., *et al.* 2006. Global variation in copy number in the human genome, *Nature* 444, 444-454.

Sharp. A.J., Locke, D.P., and McGrath, S.D., *et al.* 2006. Segmental duplications and copy-number variation in the human genome, *Am. J. Hum. Genet.* 77, 78-88.

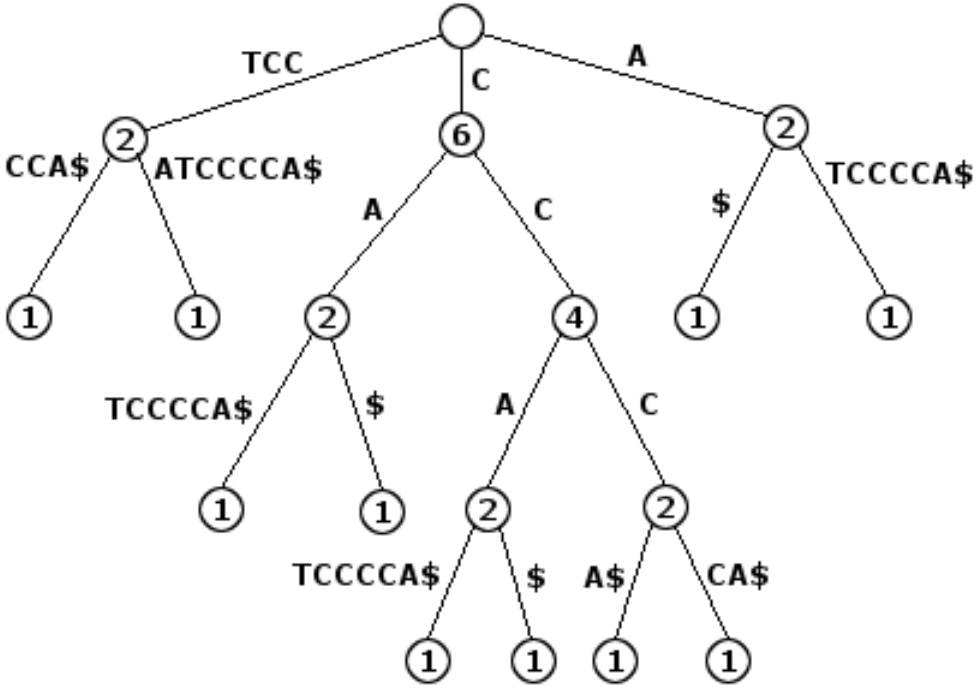
Stephen,G.A. 1994. *String searching algorithms*, World Scientific Press, New York.

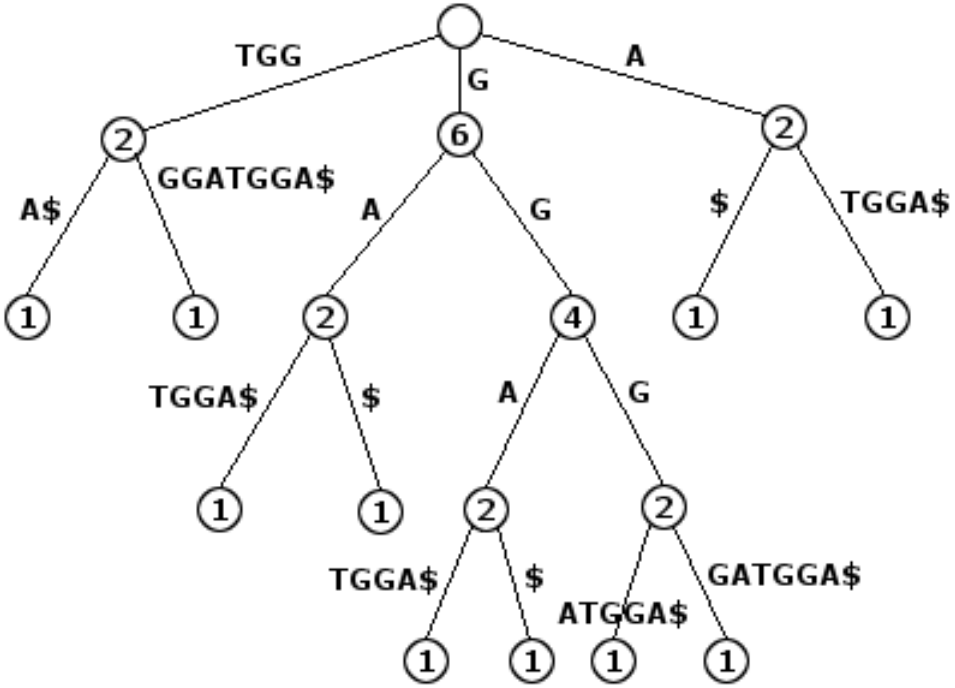
Stranger, B.E., Forrest, M.S., and Dunning, M.,*et al.* 2007. Relative impact of nucleotide and copy number variation on gene expression phenotypes, *Science* 315, 848-853.

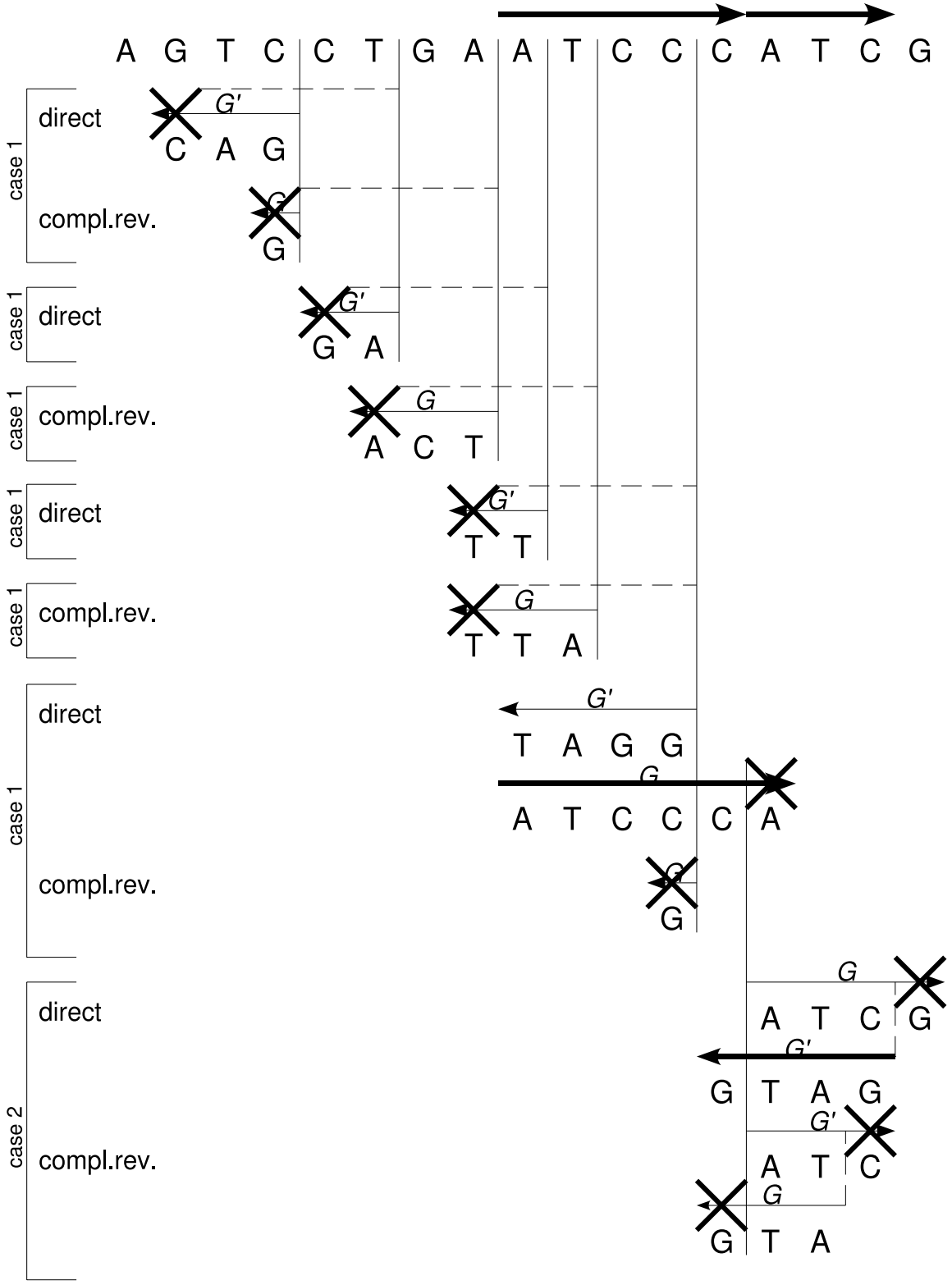
Varré, J.S., Delahaye, J.P., and Rivals, E. 1999. Transformation distances: a family of dissimilarity measures based on movements of segments, *Bioinformatics* 15, 194-202.

Weiner, P. 1973. Linear pattern matching algorithm, *14th Annual IEEE Symposium on Switching and Automata Theory*, 1-11.

Zhang, J., Feuk, L., Duggan, G.E., Khaja, R., and Scherer, S.W. 2006. Development of bioinformatics resources for display and analysis of copy number and other structural variants in the human genome, *Cytogenet Genome Res.* 115, 205-214.


TCC
C
A
2
6
2
CCA$
ATCCCCA$
A
C
$
TCCCCA$
1
1
2
4
1
1
TCCCCA$
$
A
C
1
1
2
2
TCCCCA$
$
A$
CA$
1
1
1
1


TGG
G
A
2
6
2
A$
GGATGGA$
A
G
$
TGGA$
1
1
2
4
1
1
TGGA$
$
A
G
1
1
2
2
TGGA$
$
ATGGA$
GATGGA$
1
1
1
1


A G T C C T G A A T C C C A T C G
case 1
direct
G′
C A G
case 1
compl.rev.
G
G
case 1
direct
G′
G A
case 1
compl.rev.
G
A C T
case 1
direct
G′
T T
case 1
compl.rev.
G
T T A
case 1
direct
G′
T A G G
G
A T C C C A
case 1
compl.rev.
G
G
case 2
direct
G
A T C G
G′
G T A G
case 2
compl.rev.
G′
A T C
G
G T A

sequences:

atggggctctgggcg**ctgt**tgcctggctgggtttctgctacgctgctgctggcgctggccgctctgcccgcagccctggctgcca**acag**cagtggccgatggtggggtattgt

atgaacgcccctctcggtggaatctggctctggctccctctgctcttgacctggctcacccccgaggtcaactcttcatggtggtacatgagagct**acag**gtggctcctccag

atggagccccacctgctcgggctgctcctcggcctcctgctcggtggcaccagggtcctcgctggctacccaatttggtggtccctggccctgggccagcagtacacatctct

atgaagaagtccattggaatattaagcccaggagttgctttggggatggctggaagtgcaatgtcttccaagttcttcctagtggctttggccatatttttctccttcgccca

atgctgccgcccttaccctcccgcctcgggctgctgctgctgctgctc**ctgt**gcccggcgcacgtcggcgga**ctgt**ggtggg**ctgt**gggcagcccccttggttatggaccctac

atgaaccggaaagcgcggcgctgcctgggccacctctttctcagcctgggcatggtctacctccggatcggtggcttctcctcagtggtagctctgggcgcaagcatcat**ctg**

atgc**acag**aaactttcgcaagtggattttctacgtgtttctctgctttggcgtc**ctgt**acgtgaagctcggagca**ctgt**catccgtggtggccctgggagccaacatcatctg

atggggaac**ctgt**ttatgctctgggcagctctgggcatatg**ctgt**gctgcattcagtgcctctgcctggtcagtgaacaatttcctgata**acag**gtcccaaggcctatctgac

atgtttctttcaaagcctt**ctgt**gtacat**ctgt**cttttcac**ctgt**gtcctccaactcagcc**acag**ctggtcggtgaacaatttcctgatgactggtccaaaggcttacctgat

atgctggaggagccccggccgcggcctccgccctcgggcctcgcgggtctc**ctgt**tcctggcgttgtgcagtcgggctctaagcaatgagattctgggcctgaagttgcctgg

atgagggcgcggccgcaggtctgcgaggcgctgctcttcgccctggcgctccagaccggcgtgtgctatggcatcaagtggctggcg**ctgt**ccaagacaccatcggccctggc

atgttggatggccttggagtggtagccataagcatttttggaattcaactaaaaactgaaggatccttgaggacggcagtacctggcatacctac**acag**tcagcgttcaacaa

atgctggatgggtccccgctggcgcgctggctggccgcggccttcgggctgacgctgctgctcgccgcgctgcgcccttcggccgcctacttcgggctgacgggcagcgagcc

atgcgcccccgcccgcgctggccctggccgggctctgcctgctggcgctgcccgccgccgccgcctcctacttcggcctgaccgggcgggaagtcctgacgcccttcccagg

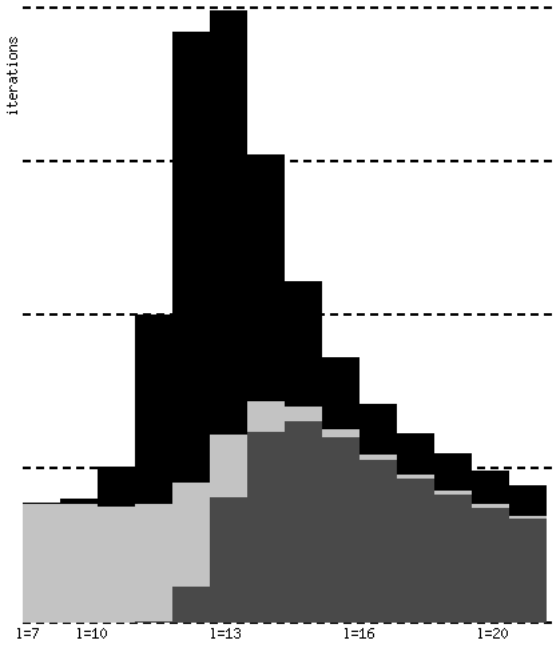
atgcagctcaccacttgcctcagggagaccctcttc**acag**gggcttctcaaaagacctccctatggtggttgggcattgcctccttcgggggttccagagaagctgggctgcgc

atgagtccccgctcgtgcctgcgttcgctgcgcctcctcgtcttcgccgtcttctcagccgccgcgagcaactgg**ctgt**acctggccaag**ctgt**cgtcggtggggagcatctc

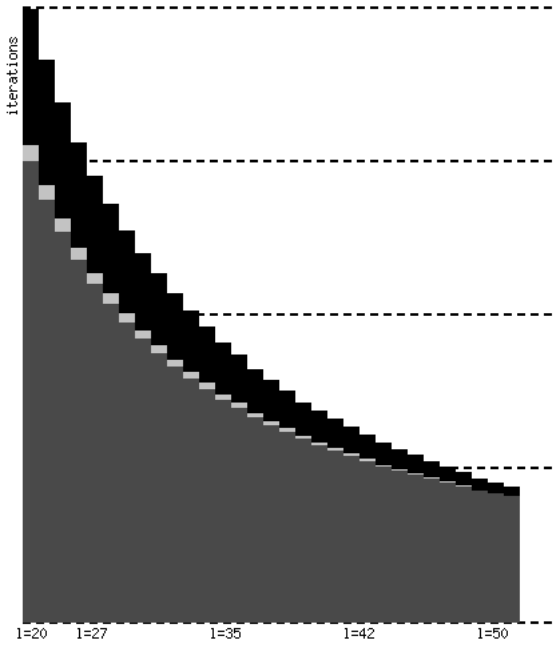
atgggcagcgcccaccctcgcccctggctgcggctccgaccccagccccagccgcggccagcgctctgggtgctc**ctgt**tcttcctactgctgctggctgctgccatgcccag

atgcccagcctgctgctg**ctgt**tcacggctgctctg**ctgt**ccagctgggctcagcttctg**acag**acgccaactcctggtggtcattagctttgaacccggtgcagagacccga

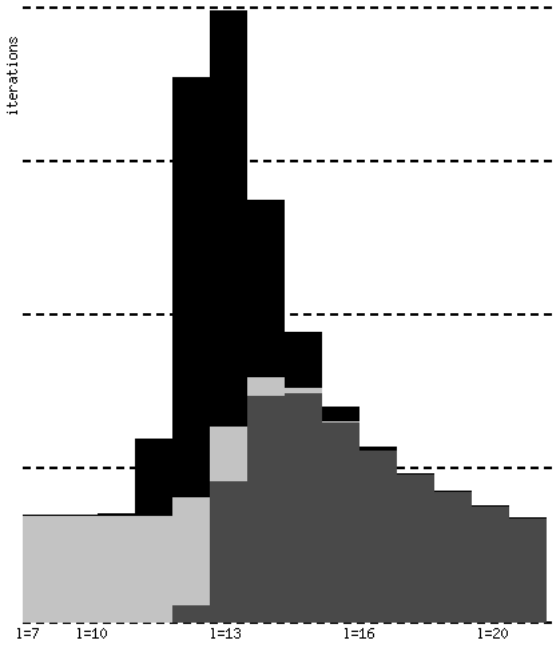
atggccccactcggatacttcttactcctctgcagcctgaagcaggctctgggcagctacccgatctggtggtcgctgg**ctgt**tgggcc**acag**tattcctccctgggctcgca



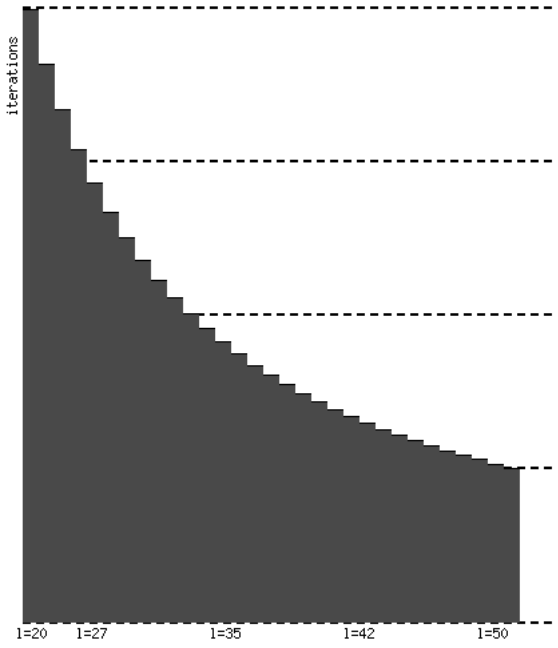
a)



b)



a)



b)