A novel approach for heuristic pairwise DNA sequence alignment

Beat Wolf, Pierre Kuonen
Department of Computer Science
University of Applied Sciences Western Switzerland, Fribourg

March 18, 2013

Abstract: *With the ever increasing speed at which DNA can be sequenced the available computing power is struggling to follow at the same pace. In contrast to earlier days of DNA sequencing, it now takes longer to align the data than to sequence it. Thus, new approaches to solve the alignment problem need to be investigated. A perfect sequence alignment might not always be required, specially when available computing power is limited. With that in mind, a purely heuristic approach to sequence alignment is proposed and evaluated. The evaluation shows good performance and quality with the used datasets, but suggest that the algorithm should not be used as the final alignment step, but to quickly identify alignment candidate locations.*

Keywords: *Algorithms, Sequence alignment, Heuristics, NGS*

Contacts: *Beat Wolf : beat.wolf@hefr.ch, Pierre Kuonen: pierre.kuonen@hefr.ch*

Conference: *BIOCOMP'13*

1 Introduction

Most current aligners use at some point in their alignment the Waterman-Smith algorithm [1], or a closely related one like Needleman-Wunsch [2] or Gotoh [3]. Examples of such an aligners are SHRiMP [4] and SSAHA [5]. Algorithms of that family will find an optimal alignment, but at the cost of computing time and memory consumption. They also struggle to adapt to biological rules of the sequences to align. Those include affine gap penalities, where gap starts and gap extensions in the sequence to be aligned are scored differently, or sequencer specific features, like homopolymer errors, where repetitions of nucleotides are sequenced in the wrong number. To address those problems, we developed a new heuristic algorithm based on a novel approach.

2 Alignment

With next generation sequencing (NGS) techniques, DNA is sequenced in small sequences, ranging from 30 to 1000 nucleotides in length. The origin of those sequences on the source genome is unknown after the sequencing process. To find their location on the original genome, the location on the reference genome is searched where the sequence has the least amount of mismatches. Those mismatches mostly come in the form of SNPs (Single Nucleotide Polymorphism, a single change between the sequence and the reference) but also indels (insertions or deletions of a certain number of nucleotides). To speed up the detection of the most likely place to which the sequence has to be aligned, many algorithms use as a starting point for the alignment a seed. The seed is a portion of the sequence to be aligned that can easily be located on the reference genome using an index. Figure 1 shows sequence aligned against a reference, starting from the initial seed TA, containing indels. The SNPs and indels are marked in blue. An alignment algorithm tries find the optimal way to extend the seed, to find the alignment with the maximal score. The score is calculated using the differences between the sequence and the reference. SNPs and indels have different costs, leading to the final alignment score.

Figure 1: Sequence alignment

Most sequences to align will have several seeds that map to the reference. Those seeds can map to regions near each other, or to regions far apart in the reference genome. Some seeds might also map to several different positions. The seeds are evaluated according to some criteria to select only the most promising ones and then align them using a classic algorithm. The evaluation of the seed can be based on its uniqueness or as in the BLAST [6] algorithm, by extending left and right, to score the seed depending on the amount of matching bases. It is on this idea of extending the seed that our novel algorithm is based. BLAST expands the seeds allowing no gaps in the extension. This is done by a simple comparison between the bases of the sequence and the reference, starting from the seed. If there is an indel close to the seed, the score will naturally be low. The proposed algorithm does the same extension, but allows for indels at every step.

3 Algorithm

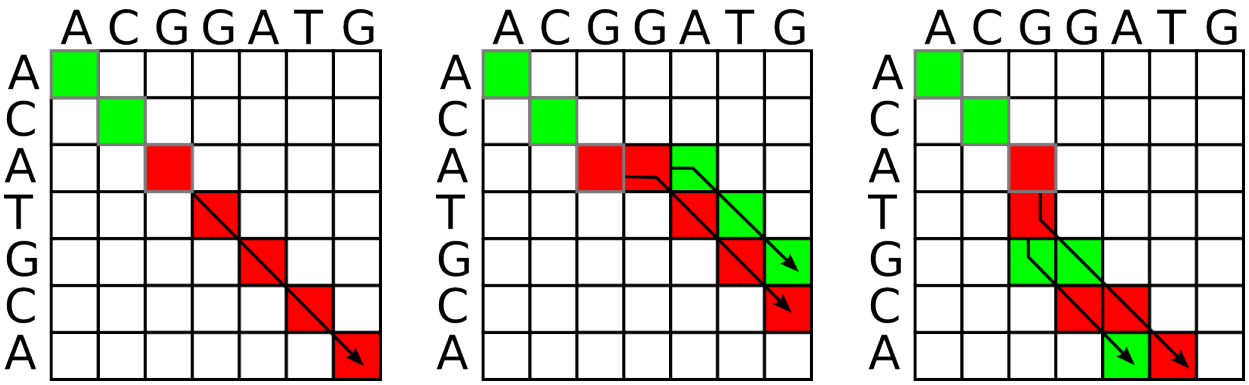
The Gotoh algorithm solves the alignment problem by creating a matrix where every position of the read is compared to every position of the reference. Using the score of the neighbor base in the matrix, a matrix containing the maximum score for the alignment at every position is created, allowing to find the optimal alignment between the two sequences. As described in [7], heuristic algorithms have a big potential of speeding up the alignment process. Several approaches already exist in this domain, ranging from evolutionary algorithms [8] to ant colonies [9]. Our algorithm takes a different approach, similar to a text file comparison algorithm developed by Miller and Myers [10], but with less memory requirements, but also less precision. Starting at a seed, first the downstream (left) and then the upstream (right) part of the sequence is aligned. As long as the sequence and the reference match, the seed is extended. As soon as a mismatch is found, a decision has to be taken, either the mismatch is a insertion, a deletion or a simple mismatch. To make this decision, all three possibilities are evaluated. To do so, the score of all possibilities is evaluated, and the best one is chosen to continue the extension. This is where the algorithm differs from other algorithms, as every decision is final and can not be reevaluated at a later stage. To evaluate every possibility, the gapless extension score (sum of all matches and missmatches from that position) is added to the base cost for that choice (a short indel will for example have a lower cost than a longer one). The extension score is based on the comparison of the sequence and the reference, but gives less weight to the positions further away from the seed than the ones closer to it.

The following paragraph will explain the algorithm trough an example. In this example the sequence ACATGCA (left) is aligned against the reference ACGGATG (top), allowing for indel sizes of maximum 2. Starting at the top left in Figure 2a, 2 matches are found. Matches are marked as green, mismatches as red. As long as the reference and the sequence to align math, the alignment can be extended with no further options beeing evaluated (as seen for the two first positions). On the third comparison from the top left, a mismatch is found.

Now all possibilities have to be evaluated. Those possibilities are:

- The mismatch is a normal mismatch and the alignment continues on the same diagonal, figure 2a
- The mismatch is a deletion of size 1 or 2, figure 2b
- The mismatch is a insertion of size 1 or 2, figure 2c

In figure 2a the mismatch case is evaluated. This is done by simply extending the alignment on the same diagonal. Figure 2b shows 2 scenarios, a deletion of 1 or 2 bases. For those scenarios, 1 or 2 bases are skipped horizontally in the matrix, and the diagonals are explored. Same goes for figure 2c where the 2 possible insertion scenarios are explored, this time skipping 1 and 2 bases vertically. Visually it can be seen that the 2 base deletion scenario is the best one, as all the bases match on the diagonal. To evaluate which option is



(a) Mismatch scenario (b) Deletion scenario (c) Insertion scenario

Figure 2: Alignment options

the best, there are several possibilities. Either the total amount of match and mismatches can be counted for every option, or matches and mismatches at the beginning of the diagonals are weigthed more. The later approach has the advantage that if there are indels on the diagonal to be tested, they will not influence the score too much. This is the approach the proposed algorithm takes. This also has the advantage that not the complete diagonal needs to be tested, as at a certain distance the importance of a comparison will be so low, that it can be discarded.

A simple linear function is used, where r is the reference sequence, s the sequence to be aligned, x the maximum amount of bases to look at in the diagonal and M the score for a matching position.

$$f(r,s) = \sum_{p=0}^{x} m(r_p, s_p) * (\tfrac{x-p}{x} * M)$$

$m(a,b)$ is a function that compare two characters:

$$m(a,b) = \begin{cases} 1 & \text{if } a \text{ equals } b \\ -1 & \text{otherwise} \end{cases}$$

To make the final call on which scenario is the best one, the base cost of the scenario is added to the score calulated with $f(r,s)$. Insertion and deletions have a higher cost in the scoring function than a simple mismatch. This leads us to the 3 functions describing the cost of every option:

$$\text{mismatch score} = f(r,s)$$
$$\text{insertion of length } l \text{ score} = I + E * (l-1) + f(r_p, s_{p+l})$$
$$\text{deletion of length } l \text{ score} = I + E * (l-1) + f(r_{p+l}, s_p)$$

Where I is the cost to start an indel, and E is the cost to increase the size of an existing indel by 1. The Option with the highest score will be chosen.

After the alignment is finished, several post processing techniques can be applied. For example trimming bad quality borders. Due to the way DNA sequencing works, the borders of the sequences are often less accurate. To address this, one of the processing steps cuts off faulty ends. That way mismatches at both ends are cut off. By default a both ends can be cut off up to 5% of the total sequence length. Other algorithms like Gotoh do this implicitly. Other improvements that can be made to the sequence is to move indels to correct small indel placement errors. Other improvements that can be made are merging of indels that are close to one another and placing indels at the leftmost position in a homopolymer.

4 Complexity

The way the insertion and deletion score is calculated allows to add affine gap penalty costs easily, in the example formula for the indel extension score it is already integrated. The algorithm is designed to abort if a certain amount of mismatches or insertions and deletions have been found. The maximum indel size can also be configured. Those configurations have a direct impact on the complexity of the algorithm. If there is a maximum of x mismatches allowed, the maximum length of the indels is l and the maximum length of the diagonals to be explored is 16, then the maximum amount of comparisons in the algorithm is $m + 2xl * 16$, which translates to a complexity of $O(m+xl)$, where m is the length of sequence to be aligned. The minimum complexity, if the reference and the sequence are a perfect match, is $O(m)$. Compared to the complexity of at least $O(n * m)$, where n is the length of the reference sequence, of the classical algorithms, the proposed algorithm is potentially faster than the classical ones, specially if the initial seed is chosen well. It has to be noted, that n is usually reduced to be similar in size to m when aligning, which leads to a complexity of $O(m^2)$.

Assuming that the complete reference and sequence can already be found once in the memory and as there is no matrix to be constructed, the memory complexity is $O(1)$. The size of the reference and the size of the sequence to be aligned do not affect the memory usage as the algorithm only solves one mismatch at a time and takes a final decision on how to align it, this keeps the memory requirement low. This is very good compared to the classical algorithms which often have a memory complexity of $O(n * m)$, or $O(m^2)$ when n is similar in size to m. The Gotoh algorithm for example creates 3 matrices, each of the size $m * n$.

5 Quality

To test the alignment quality, 40k simulated sequences where aligned using the heuristic approach and the Gotoh algorithm, which is an improved version of the Waterman-Smith algorithm, allowing for affine gaps while keeping the same execution and memory complexity. The correct seed for every sequence was known, thus both algorithms had to check only one candidate position, which should result in a succesful alignment. The sequences contained up to 5% errors and on top of that up to 3% deletions with a length of 1 to 5. Two datasets where produced, one with only 2 deletions which can not be close together, and one where a variable amount of deletions can be found, potentially close together. Figure 3 shows the boxplot for the first dataset and the second dataset. The Figures contain the boxplot of the percentage of the score the heuristic algorithm achieved compared to the Gotoh algorithm. The result is rather good for the heuristic algorithm, specially in the case of only 2 deletions per sequence. In the second dataset where there is a variable amount of deletions per sequence, one of the weaknesses of the algorithm appears. When 2 deletions are close together, the algorithm can take the wrong decision and decide to use a insertion instead of a deletion. But even considering that, the median score is about the same as the one achieved by Gotoh. Both dataset show outliners, with scores as low as 3% and 20% worse than the Gotoh algorithm.

Without looking at the outliners we can see that the alignment using the heuristic is nearly identical to the Gotoh version on the first dataset. It even scores slightly higher in certain cases, which is due to a more aggressive skipping behavior of Gotoh, which can lead to too some unnecessarily skipped bases at the borders of the sequence. Further testing revealed one quality problem with the heuristic approach. Depending on which seed is chosen in a sequence, the results can slightly vary. While they may be correct (for example an indel that can potentially be placed at different places with the same score), the variant detection on the alignment becomes harder.

It has also to be noted, that for the second dataset 2.4% of the sequences did not find a valid alignment using the heuristic approach.

6 Performance

To assert the performance of the algorithm, it was directly compared to the Gotoh algorithm. Both algorithms where implemented in a unaccelerated version, meaning, no multithreading, GPU (Graphics processing unit) acceleration or SIMD(Single Instruction, Multiple Data) instructions where used that could affect both algorithms differently. There is a big potential to speedup the Gotoh algorithm using those techniques

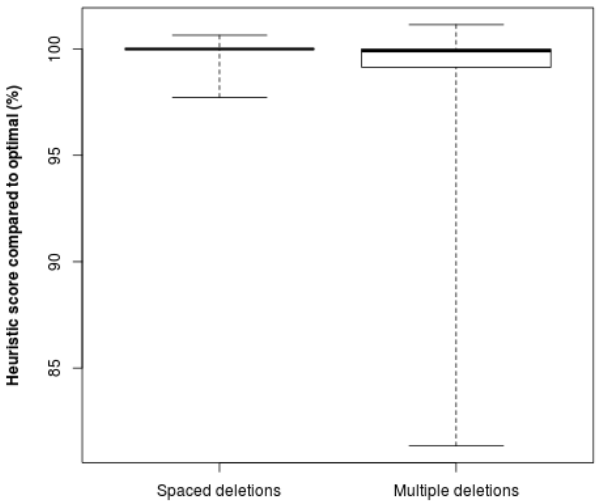


Figure 3: Score (in %) achieved by the heuristic algorithm compared to Gotoh on 2 datasets

[11], while this potential is currently not explored for the proposed algorithm. The second dataset, which has more deletions, was used. Figure 4 shows the result of this comparison (logarithmic scale), which is as expected by the complexity of both algorithms. The time needed to align using the Gotoh algorithm increases quadratically with the read size, but with the proposed algorithm it increases only linearly. For sequences of length 100, the heuristic is about 6 times faster than Gotoh and for sequences of length 350 about 24 times faster.

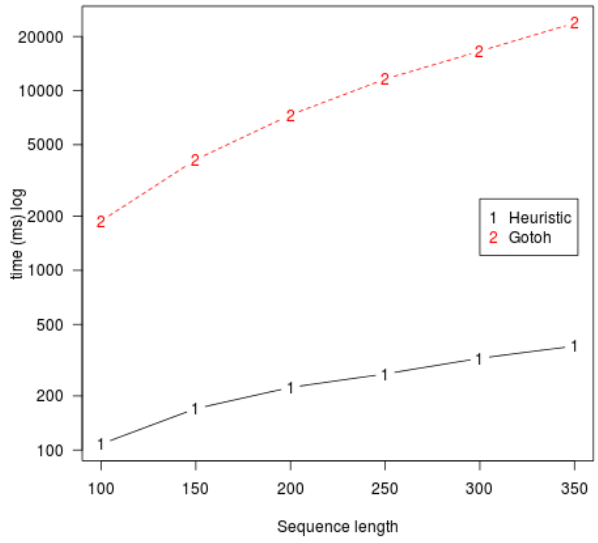


Figure 4: Alignment time comparison

7 Possible improvements

As expected, the algorithm does not return the optimal solution for an alignment, but it does rapidly give an approximate alignment that in most cases is very close to the optimal solution. But there is a big potential

for improvements. For example, the formula described to calculate the score for an insertion or deletion does not work for long indels. For an indel to be selected as an option over a mismatch, the score for it must be higher than the mismatch case. If the base cost of the indel $(I + E * (l - 1))$ is higher than the maximum score it can get trough the extension $(\sum_{p=0}^{x} m(r_p, s_p) * (\frac{x-p}{x} * M))$, it will never be selected. With $I = 9$, $E = 2$, $M = 3$ and $x = 16$, the maximum length of an indel that can be detected (if all the bases match after the indel) is 22, but only if the Match case has the lowest possible score (-25.5) and the indel gets the highest extension score 25.5. One way to solve this problem is to make x depend on the desired maximum indel length l.
Another issue is that to choose an option, indels are no longer considered when creating the extension score, only matches and mismatches are counted. If an indel is close to the start of the diagonal to be tested, the score will be wrong and a wrong decision can be taken. This could be addressed by reapplying the same logic as in the extension during first step on every mismatch, thus creating a tree of possibilities. The debt of that tree could then be configurable, to be able to control the impact on the algorithmic complexity.
Another problem is, that when the heuristic takes a wrong decision, it will not recover from it. This problem could be limited by using multiple seeds for the same sequence, thus limiting the damage that can be done with one wrong decision.

8 Conclusion

The heuristic alignment approach is an interesting and fast one. The quality of the alignment, while expectedly not as good as with Gotoh, is very good depending on the dataset. While depending on the quality requirements, the alignment can be used directly, it is advisable to realign the sequences using an algorithm like Gotoh and only use the heuristic approach to identify good candidate positions for the final alignment. First tests to use this hybrid approach are promising and will be subject of further research. There are still many improvements can be made to the algorithm as described in 7. If the proposed improvements are viable will also be subject of further research.

References

[1] Smith, Temple F.; and Waterman, Michael S. (1981)."Identification of Common Molecular Subsequences".Journal of Molecular Biology147: 195197.doi:10.1016/0022-2836(81)90087-5.PMID7265238.

[2] Needleman, Saul B.; and Wunsch, Christian D. (1970)."A general method applicable to the search for similarities in the amino acid sequence of two proteins".Journal of Molecular Biology48(3): 44353.doi:10.1016/0022-2836(70)90057-4.PMID5420325.

[3] O. Gotoh:An improved algorithm for matching biological sequences. In:Journal of Molecular Biology. 162,1982, S.705-708

[4] Rumble, S. M., Lacroute, P., Dalca, A. V., Fiume, M., Sidow, A., & Brudno, M. (2009). SHRiMP: accurate mapping of short color-space reads. PLoS computational biology, 5(5), e1000386. doi:10.1371/journal.pcbi.1000386

[5] Ning, Z., Cox, A. J., & Mullikin, J. C. (2001). SSAHA : A Fast Search Method for Large DNA Databases, (2), 17251729. doi:10.1101/gr.194201.1

[6] Altschul, S; Gish, W; Miller, W; Myers, E; Lipman, D (October 1990)."Basic local alignment search tool".Journal of Molecular Biology215(3): 403410.doi:10.1016/S0022-2836(05)80360-2.PMID2231712.

[7] Bucak, I. Ö., & Uslan, V. (2010). An analysis of sequence alignment: heuristic algorithms. Conference proceedings : Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference, 2010, 18247. doi:10.1109/IEMBS.2010.5626428

[8] C. Zhang, A.K. Wong, "A genetic algorithm for multiple molecular sequence alignment", Comput. Applic. Biosci., Vol. 13, pp. 565-581, 1997.

[9] W. Chen, B. Liao, W. Zhu, H. Liu, Q. Zeng, "An ant colony pairwise alignment based on the dot plots", Journal of Computational Chemistry, Vol. 30, pp. 93-97, 2008.

[10] Miller, W., and Myers, E.W. 1985. A file comparison program. SoftwarePractice and Experience 15, 10251040.

[11] Rognes, T. (2011). Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation. BMC bioinformatics, 12(1), 221. doi:10.1186/1471-2105-12-221