BIOINFORMATICS

Vol. 00 no. 00 2005
Pages 1–2

Seeq: a library for inexact DNA sequence matching

Eduard Zorita [1,2] and Guillaume J. Filion [1,2*]

[1]Genome Architecture, Gene Regulation, Stem Cells and Cancer Programme, Centre for Genomic Regulation (CRG), Dr. Aiguader 88, 08003 Barcelona, Spain
[2]Universitat Pompeu Fabra (UPF), 08002 Barcelona, Spain

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT
Motivation: Searching specific sequences over DNA reads is an elemental procedure of biological data analysis. Experimental DNA samples have usually been engineered to contain pattern sequences at the positions where operations such as adaptor removal, sequence trimming or barcode extraction have to be performed. The experimental data contain errors that must be tolerated in order to identify these patterns at their correct positions. This process is straightforward in data produced by second generation sequencing machines, in which the error rates lie below 2%. Short sequences with small error tolerance suffice to identify the patterns accurately in these circumstances. However, third generation sequencing technologies promise longer reads with error rates around 15%. Consequently, this opens a new scenario in which both the pattern length and the relative error tolerance will have to be dramatically increased.
Results: In this paper we present seeq, an inexact pattern matching algorithm optimized for DNA sequence matching with almost constant running time. We benchmark seeq against the most widely used algorithms for inexact pattern matching. The results show that seeq scales better and becomes significantly faster as the error rate increases.
Availability: Seeq is available in three forms: Linux software, C Library and Python module. The source code and the C/Python libraries are available for download at http://github.com/ezorita/seeq.
Contact: guillaume.filion@gmail.com

1 INTRODUCTION

Inexact string matching is an old problem that dates back to the sixties with the definition of the edit distance (Levenshtein, 1965). The first algorithms used to solve this problem were based on dynamic programming (Needleman and Wunsch, 1970). Although the algorithms were not very efficient, they are still widely used to align biological sequences due to their simplicity and flexibility. Later on, more sophisticated algorithms derived from the Boyer-Moore search method (Boyer and Moore, 1977) were designed to search patterns on long strings allowing differences. Such algorithms are much faster than the dynamic programming approach because they do not compute all the relative distances. Instead, the pattern is pre-processed to design a search strategy which skips parts of the text that do not contain any match. The jumping strategy is very efficient for perfect matches or small edit distances because the non-matching condition is inferred with only a few comparisons. In this scenario, the number of operations per input character is ridiculously small given that large areas of the text are never processed. However, other scenarios are not so favorable for this technique. For instance, if the proposed pattern matches in almost any text position, the performance degrades severely since the algorithm will not be able to jump so often. Besides, for high mismatch ratios the search strategy becomes complex and requires more comparisons to determine a non-matching condition, specially when the alphabet is as small as in the task of matching DNA sequences.

Sequencing data produced by the Illumina platform (Margulies *et al.*, 2005) typically has 1-2% error rate mainly consisting of substitutions (Dohm *et al.*, 2008). Therefore one expects to find all the matching strings within a small edit distance relative to the original pattern length. Moreover, if such patterns are produced artificially (e.g. sequencing adaptors or barcodes), it is reliable to use short sequences that are not present in the target genome. On the other side, the PacBio platform produces reads two orders of magnitude longer than Illumina, but at the expense of a much higher error rate (15%) consisting principally of insertions and deletions (Eid *et al.*, 2009). This high error rate dramatically increases the false positive rate unless the pattern sequence is extended accordingly. This brings us to a new scenario with unusually long patterns and edit distances that lie on the same order of magnitude.

In this article we present *seeq*, an algorithm based on a deterministic finite automaton (DFA) structure. The automaton is built as the input text is processed and has been optimized for DNA sequence matching. We benchmark seeq against other widely used inexact string matching software. The results show that seeq scales better and becomes significantly faster as the error rate increases.

2 METHODS

The algorithm proposed is the DFA representation of a dynamic programming algorithm. The goal is to find the occurrences of a pattern P in a text T that are k-differences or less. The lengths of P and T are m and n, respectively. The score matrix $\mathbf{C} \in \mathbb{N}^{m\times n}$ is initialized so as that any text position is a potential alignment start, i.e. $\mathbf{C}(0,j) = 0$ and $\mathbf{C}(i,0) = i$. After the initialization, the score matrix is computed and filled column-wise. Note that the standard dynamic programming update process can be represented as $\mathbf{c}_j = f(\mathbf{c}_{j-1}, T_j)$. In effect, the value of the j-th column $\mathbf{c}_j$ is a deterministic transition that only depends on $\mathbf{c}_{j-1}$ and the character T_j. We take advantage of this property to build a deterministic

*to whom correspondence should be addressed

Fig. 1.

automata in which the different states represent the set of non-equivalent alignment columns and the transitions between states are dictated by the input characters.

To make the alignment algorithm efficient, only the significant ($\mathbf{C}(i,j) \leq k$) parts of the column are computed. The DFA states S have a vector representation $\mathbf{s} = \text{dif}(\mathbf{c}_j)$ generated by differentially encoding the j-th column values as $\mathbf{s}(i) = \mathbf{C}(i+1,j) - \mathbf{C}(i,j) + 1$ for $i = 0 \ldots m-1$. This is done for two reasons: (i) the alphabet of $\mathbf{s}$ is small $\{0,1,2\}$, therefore the alignment columns can be efficiently stored in a ternary trie, and (ii) the inverse process to recompute the original column ($\mathbf{c}_j = \text{undif}(\mathbf{s})$) is performed without further information, provided that $\mathbf{C}(0,j) = 0$.

The DFA construction process is illustrated in Figure. X. The dynamic programming algorithm starts at the DFA state $S = S_0$. For any input character T_j, being S the active DFA state, the algorithm recursively proceeds as follows:

- If S has an outgoing path (to S_q) with label T_j, then $S = S_q$.
- Otherwise, the vector $\mathbf{s}_j$ is computed as $\mathbf{s}_j = \text{dif}(f(\text{undif}(\mathbf{s}), T_j))$. If a state represented by $\mathbf{s}_j$ already exists (S_q), S is connected to such state with a path labeled T_j and $S = S_q$. On the other hand, if $\mathbf{s}_j$ does not exist, a new DFA state S_j is created and both nodes are connected in the same way, then $S = S_j$.

After several steps, the DFA graph is densely connected and the complexity of the algorithm boils down to direct DFA state transitions.

3 RESULTS

We have benchmarked seeq against two competitive inexact string matching algorithms: agrep (Wu and Manber, 1992) and nr-grep (Navarro, 2001). We used two real sequencing datasets for benchmarking. Both datasets contain 10 million reads but differ in read length. Dataset 1 consists random Drosophila genome reads of length 100. Dataset 2 reads are 50 nucleotides long, from which the first 30 nucleotides are constant and the last 20 are random (barcodes). The software were set to match random patterns ranging from 5 to 50 nucleotides with a random error tolerance up to 40%. We would like to point out that agrep limits the number of differences to 8, therefore the comparison shown here is only fair between seeq and nr-grep.

Figure 2 shows the running time of each execution as a function of the relative error tolerace (k/m). For low error rates, both agrep and nr-grep perform better than seeq, with a maximum 5-fold advantage when $k = 1$ and $m > 25$. On the other hand, seeq is up to two orders of magnitude faster than nr-grep and one order of magnitude faster than agrep for $k/m > 0.15$. The similarity between the two datasets indicates that neither software is particularly favored when the processed data has a repetitive structure. The worst to best case ratio also speaks strongly in favor of seeq (5), which achieves almost constant running time. This value is one and almost three orders of magnitude higher for agrep (50) and nr-grep (1000), respectively.

4 CONCLUSION

In this article we have introduced and benchmarked seeq, an inexact DNA sequence matching algorithm. We have measured and compared the running time of seeq and two competitive software in different configurations. The results conclude that seeq is faster in higher error regions with a worst case speedup of almost 100. We attribute the higher efficiency of seeq to the fact that it has been specifically designed to match DNA sequences. Thus, we could take advantage of algorithms that are not feasible in general string matchers because they only scale well when the size of the alphabet is reduced.

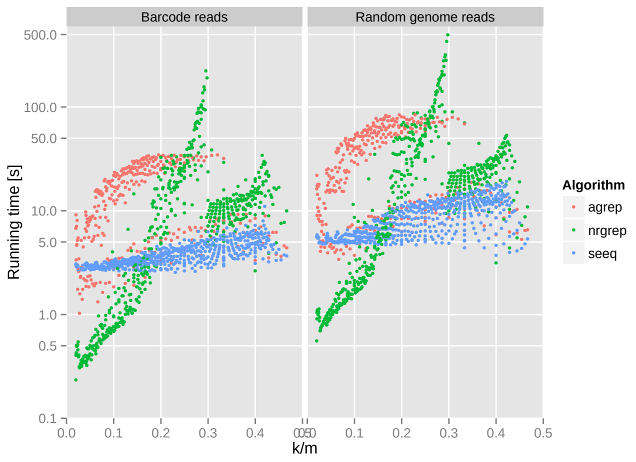

Fig. 2. Benchmark between seeq, agrep and nr-grep.

Funding:

REFERENCES

Boyer, R. S. and Moore, J. S. (1977). A fast string searching algorithm. *Commun. ACM*, **20**(10), 762–772.

Dohm, J. C., Lottaz, C., Borodina, T., and Himmelbauer, H. (2008). Substantial biases in ultra-short read data sets from high-throughput dna sequencing. *Nucleic acids research*, **36**(16), e105–e105.

Eid, J., Fehr, A., Gray, J., Luong, K., Lyle, J., Otto, G., Peluso, P., Rank, D., Baybayan, P., Bettman, B., *et al.* (2009). Real-time dna sequencing from single polymerase molecules. *Science*, **323**(5910), 133–138.

Levenshtein, V. I. (1965). Binary codes capable of correcting spurious insertions and deletions of ones. *Problems of Information Transmission*, pages 1:8–17.

Margulies, M., Egholm, M., Altman, W. E., Attiya, S., Bader, J. S., Bemben, L. A., Berka, J., Braverman, M. S., Chen, Y.-J., Chen, Z., *et al.* (2005). Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, **437**(7057), 376–380.

Navarro, G. (2001). Nr-grep: a fast and flexible pattern-matching tool. *Software: Practice and Experience*, **31**(13), 1265–1312.

Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, **48**(3), 443 – 453.

Wu, S. and Manber, U. (1992). Agrep–a fast approximate pattern-matching tool. *Usenix Winter 1992*, pages 153–162.