

Space-efficient whole genome comparisons with Burrows-Wheeler transforms

Ross A. Lippert,¹

Keywords: string-index, suffix, Burrows-Wheeler transform, genomics

1 Introduction.

The starting point for any alignment of mammalian genomes is the computation of exact matches satisfying various criteria. Time-efficient, $O(n)$, data structures for this computation, such as the suffix tree[7], require $O(n \log(n))$ space, several times the space of the genomes themselves. Thus, any reasonable whole-genome comparative project find itself requiring a tens of Gigabytes of RAM to maintain time-efficiency. This is beyond most modern workstations.

With a new data-structure, the compressed suffix array (CSA) implemented via the Burrows-Wheeler transform[2], we can trade time-efficiency for space-efficiency, taking $O(n \log(n))$ time, but running in $O(n)$ space, typically in total space less than or equal to that of the genomes themselves. If space is more expensive than time, this is an appropriate approach to consider.

An implementation (called *bbwt*) was demonstrated by aligning two mammalian genomes on a modest workstation equipped with under 2 GB of free RAM in time superior to that of the implementations of other data structures.

2 Compressed Suffix Arrays.

A compressed suffix array [4, 3] can be implemented in terms of a Burrows-Wheeler transform (BWT) string using population counts of prefixes to compute indices. Implementations exist supporting these queries efficiently which can be stored in $O(n \log |\Sigma|)$ bits as well as constructed in $O(n|\Sigma| \log(n))$ time [5]. On a nucleotide alphabet this translates to 5 bits per character to store and 10 bits (worst case) during construction. This author, with collaborators, has developed a similar implementation[6] requiring only 2.5 bits per character to store and 5 bits (worst case) during construction.

The BWT itself allows one to query the number of occurrences of a pattern P in S in $O(|P|)$ time. Here we address the question of whether we can use this data-structure to do the sort of query a comparative genomics effort might require and how it compares to more familiar data-structures.

3 Implementation and Experiments.

We used our implementation to find all bi-unique 20-mers in common between the human and mouse genomes available from NCBI. The source for these routines is being made available².

The two assembly files were downloaded from ftp://ftp.ncbi.nih.gov/genomes/H_sapiens and ftp://ftp.ncbi.nih.gov/genomes/M_musculus on August 17, 2004. The files are slightly

¹Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, Earth. E-mail: lippert@math.mit.edu

²<http://www-math.mit.edu/~lippert/software/bbwt/>

less than 2.84 GB and 2.74 GB in size, respectively. Additionally, the file `NCBIhs-rc.fasta` was produced, containing the reverse complements of the strings in `NCBIhs.fasta`.

The programs were run on a Macintosh G5 with a 1.8 GHz PowerPC 970 processor with 1.8 GB of free RAM (the operating system would not allow more to be allocated, though the hardware had 2.5 GB). The programs were compiled with `GCC` version 3.3 with the command and options `g++ -g -O4`.

The total time to produce the indices for the three genome files was *30h12m*. The total runtime to produce the matches from the BBWTs was *14h22m*.

A suffix tree on a sequence of length n can be built and searched in approximately Tn time and Sn space. With REPuter, on our platform, T and S can be determined empirically by constructing a suffix tree on a sufficiently large sequence (we used prefixes of mouse chromosome 1). The constants were determined to be $T = 4.25 \times 10^{-6}s$ and $S = 12.7$. The computation of common 20-mers between a pair of strings x and y is typically accomplished by building a suffix tree on $x\%y$ where $\%$ is a unique character. We can extrapolate that this would require 70 GB of space and *6h30m* of time for each of the orientations.

A plausible means to fit a computation of this size within 1.8 GB, is to partition the problem into smaller pieces and compute all 20-mer matches among the pieces (applying the count criteria is a complicating issue, but this is an optimistic analysis). In dividing the problem into 39×39 sub-problems, the run-time is increased by a factor of 39 to *256h* per strand or *512h* total.

The more recent match finder `vmatch` (available from <http://www.vmatch.de>) is based on enhanced suffix arrays [1] and has better time and space characteristics than REPuter (about $5n$ bytes per character). We again can extrapolate a runtime based on the runtime of a problem $\frac{1}{14}$ th the size, which would just fit within our memory limits. The runtime of the fractional problem was measured to be *19m30s*, which, when scaled by $2 \times 14 \times 14$ gives a total runtime of *130h* for both strands.

4 References and bibliography.

References

- [1] M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch. The enhanced suffix array and its applications to genome analysis. In *Proceedings of the Second International Workshop on Algorithms in Bioinformatics*, pages 449–463. Springer-Verlag, 2002.
- [2] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital SRC, 1994. Research Report 124. 10th May 1994.
- [3] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In IEEE, editor, *Proc. 41st IEEE Symposium on Foundations of Computer Science (FOCS '00)*, pages 390–398, 2000.
- [4] R. Grossi and J. S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching (extended abstract). In *Proc. 32nd ACM Symp. on Theory of Comp. (STOC '00)*, pages 397–406. ACM Press, 2000.
- [5] T. W. Lam, K. Sadakane, W.-K. Sung, and S.-M. Yiu. A space and time efficient algorithm for constructing compressed suffix arrays. In O. Ibarra and L. Zhang, editors, *Computing and Combinatorics: 8th Ann. Int. Conf., COCOON 2002*, volume 2387 of *Lecture Notes in Computer Science*, pages 401–410. Springer-Verlag (Heidelberg), 2002.
- [6] R. A. Lippert, C. M. Mobarry, and B. P. Walenz. A space-efficient construction of the Burrows Wheeler transform for genomic data. *submitted to J. Comp. Biology*, <http://www-math.mit.edu/~lippert/research/JCB-submission-2.pdf>, 2004.
- [7] P. Weiner. Linear pattern matching algorithms. In *Proceedings of the 14th IEEE Symposium on Switching and Automata Theory*, pages 1–11, 1973.