2006

Application of a superword array in genome assembly

Xiaoqiu Huang
Iowa State University

Shiaw-Pyng Yang
Washington University School of Medicine in St. Louis

Asif T. Chinwalla
Washington University School of Medicine in St. Louis

LaDeana W. Hillier
Washington University School of Medicine in St. Louis

Patrick Minx
Washington University School of Medicine in St. Louis

See next page for additional authors

Published online January 5, 2006

Nucleic Acids Research, 2006, Vol. 34, No. 1 ***201–205***
doi:10.1093/nar/gkj419

Application of a superword array in genome assembly

Xiaoqiu Huang*, Shiaw-Pyng Yang[1], Asif T. Chinwalla[1], LaDeana W. Hillier[1], Patrick Minx[1], Elaine R. Mardis[1] and Richard K. Wilson[1]

Department of Computer Science, Iowa State University, Ames, IA 50011, USA, and [1]Genome Sequencing Center, Washington University School of Medicine, St Louis, MO 63108, USA

Received October 3, 2005; Revised November 9, 2005; Accepted December 13, 2005

ABSTRACT

We introduce a data structure called a superword array for finding quickly matches between DNA sequences. The superword array possesses some desirable features of the lookup table and suffix array. We describe simple algorithms for constructing and using a superword array to find pairs of sequences that share a unique superword. The algorithms are implemented in a genome assembly program called PCAP.REP for computation of overlaps between reads. Experimental results produced by PCAP.REP and PCAP on a whole-genome dataset show that PCAP.REP produced a more accurate and contiguous assembly than PCAP.

INTRODUCTION

To compute quickly overlaps between reads, existing genome assembly programs locate every pair of reads that share one or more unique words of length *w* and compute an overlap between the reads in the pair (1–8). A word of length *w* is highly repetitive if the number of its occurrences in the reads is greater than a cut-off and is unique otherwise. The word length *w* is a constant number between 12 and 32. For example, the word length is 32 for the Atlas program, 24 for the Arachne program, 20 for Celera Assembler, 17 for the Phusion program and 12 for the PCAP program, where PCAP uses two close 12-base word matches to tolerate an insertion/deletion sequencing error between the word matches. Using the constant word length is less effective at finding true overlaps between repetitive reads. For instance, a pair of repetitive reads share no unique words of length 24, but share a unique superword of length 72. A superword is a segment whose length is a multiple of *w*. In other words, a superword is obtained by concatenating one or more words. Because a data structure to be presented later requires that the words be of the same length, superwords are introduced to represent segments that are longer than words.

Using superwords of variable sizes for seeding overlaps leads to an overlap computation method that is sensitive and specific in unique regions, and specific in repetitive regions. Since short superwords occur with low frequencies in unique regions, using short unique superwords allows the method to achieve a high level of sensitivity at an acceptable level of specificity. On the other hand, in repetitive regions, short superwords occur with extremely high frequencies and hence it is not efficient to use short superwords for seeding overlaps in those regions. Some repetitive regions contain long superwords with low frequencies. Thus, using long unique superwords allows the method to maintain an acceptable level of specificity in those regions.

Unique superwords in a set of reads are found by using a superword array for the set of reads. A combined sequence is obtained by concatenating the sequences of the reads with a special symbol at every read sequence boundary. The concatenation allows each read position to be represented by a unique number, which is the location of the read position in the combined sequence. A superword array for the set of reads is an array of positions of the combined sequence that are lexicographically sorted by the superwords starting at the positions, where the special boundary symbol ranks before the characters in the read alphabet.

The superword array is introduced to inherit desirable features of the lookup table (9–11) and the suffix array (12). The lookup table is easy to code and can be extended to tolerate base mismatches in word matches (13). The suffix array is effective at finding long exact segment matches (14,15).

We describe simple algorithms for constructing and using a superword array to find pairs of reads that share a unique superword. The algorithms are implemented in a genome assembly program called PCAP.REP for computation of overlaps between reads. Experimental results produced by PCAP.REP and PCAP on a whole-genome dataset show that PCAP.REP produced a more accurate and contiguous assembly than PCAP. The PCAP.REP program is freely available for academic use at http://seq.cs.iastate.edu.

*To whom correspondence should be addressed at Department of Computer Science, Iowa State University, 226 Atanasoff Hall, Ames, IA 50011-1040, USA. Tel: +1 515 294 2432; Fax: +1 515 294 0258; Email: xqhuang@cs.iastate.edu

Sequence	C	A	T	C	G	T	G	A	#	A	T	C	G	T	G	T	#
Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Word code	4	3	13	6	11	14	8	-1	-1	3	13	6	11	14	11	-1	-1
Superword array	8	9	16	17	2	10	1	4	12	7	15	5	13	3	11	6	14
Superword code	-1	-1	-1	-1	3	3	4	6	6	8	11	11	11	13	13	14	14
	3	13	-1	-1	6	6	13	14	14	-1	-1	8	11	11	11	-1	-1
	6	11	-1	-1	14	14	11	-1	-1	13	-1	-1	-1	8	11	3	-1

Figure 1. An example superword array for a combined sequence of two short reads. The combined sequence is given on the top row. For each base of the combined sequence, the position of the base is shown below the base on the second row and the code of the word of length 2 starting at the position is shown below the position on the third row. The ordered sequence positions in the superword array are shown on the fourth row. For each sequence position in the superword array, the code of the superword at word level 3 starting at the position is given as a column of three word codes on the bottom three rows.

MATERIALS AND METHODS

Definition of a superword array

We define a superword array for a set of reads as follows. Each read sequence consists of characters from the alphabet $\Sigma = \{A, C, G, T, N\}$. Let *Cseq* denote the concatenation of all read sequences in the set with a special boundary character (denoted #) inserted at every sequence boundary. Let n be the length of *Cseq*, where the positions of *Cseq* are numbered $1, 2, 3, \ldots, n$.

A word of length w is a string of w characters. The code of a word free of the characters N and # is obtained by converting the word into a base-4 number of w digits and converting the base-4 number into a decimal number (16). The code of a word with N or # is -1. Two words form an exact match if and only if the two words have the same code and the code is non-negative. Thus, two words with the same code of -1 are not considered as a match. Let $Code\,(p)$ denote the code of a word starting at position p of *Cseq*. Assume that $Code\,(p) = -1$ for any $p > n - w + 1$.

A superword at word level h is a string of length $h \times w$ that is obtained by concatenating h words. The code of a superword at word level h starting at a position p of *Cseq* is a tuple of h components, $\langle Code\,(p),\ Code\,(p + w),\ Code\,(p + 2w), \ldots, Code\,(p + (h - 1) \times w)\rangle$. For any k with $1 \leqslant k \leqslant h$, the k-th component code of the position p is $Code\,(p + (k - 1) \times w)$. A lexicographic order of superwords at word level h is defined to be a lexicographic order of their codes with the following rules for resolving the order of superwords with the same code or with component code of -1 at the same rank.

For two superwords at word level h from different positions of *Cseq*, if the superwords have the same code with non-negative component code at each rank, then the order of the superwords is based on their positions in *Cseq*. If the superwords have the same component code from rank 1 to rank k with k being the smallest rank such that the component code at rank k is -1, then the order of the superwords is based on their positions in *Cseq*. For example, a superword of code $\langle 3, 2, 4\rangle$ from position 5 of *Cseq* is before a superword of code $\langle 3, 2, 4\rangle$ from position 13 of *Cseq*, and a superword of code $\langle 3, -1, 4\rangle$ from position 37 of *Cseq* is after a superword of code $\langle 3, -1, 8\rangle$ from position 25 of *Cseq*.

A superword array *SW* for the set of reads is an array of the n positions of *Cseq* that are sorted based on a lexicographic order of the superwords at word level *wlcut* starting at the positions. The parameter *wlcut* is a cut-off on the number of word levels. Figure 1 shows an example superword array for a combined sequence of two short reads.

Construction and use of a superword array

The array *SW* is constructed by a simple method. In this method, *SW* [i] is initialized to i for each i from 1 to n. Then *SW* is sorted at word level *wlev* for each *wlev* from 1 to *wlcut*. The sorting at word level *wlev* is performed by using a lookup table. The details of the method are given in Supplementary Data.

The whole dataset is partitioned into a number of subsets of similar sizes (5). Every subset is assigned to a separate processor for computation of overlaps between reads in the subset and reads in the whole dataset. The subset is kept in the main memory of the processor, whereas the whole dataset is kept on the disk. A superword of a read from the whole dataset is unique with respect to the subset if the number of its occurrences in the subset is at most *pncut*. The parameter *pncut* is a cut-off on the number of occurrences of every unique superword. For every read f from the whole dataset, every read g in the subset that shares a unique superword with the read f is located for computation of an overlap between the reads f and g.

A superword array *SW* is constructed for the subset and kept in the main memory of the processor. The array *SW* is used to locate the occurrences of the superwords from the whole dataset in the subset as follows. The reads in the whole dataset are processed one at a time. The current read is compared with the subset by using the array *SW* for the subset. The positions of the current read are considered one at a time in increasing order. Let q be the current position. If a superword starting at the position q at a word level between 1 and *wlcut* is free of the characters N and #, and is unique with respect to the subset, then the minimum word level *wlev* and a section of *SW* are computed such that the section contains the positions of all unique occurrences of the superword at word level *wlev* starting at the position q. Otherwise, the empty section is reported for the position q. The details on using the array *SW* are given in Supplementary Data.

A special feature of the methods given above is that the methods are easy to implement because of the use of a lookup table. The use of a lookup table by our method in construction of a superword array is related to the Manber–Myers algorithm (12) for constructing a suffix array. In the Manber–Myers algorithm, a lookup table is used once to sort all segments

of variable lengths by their leftmost word whereas in our method, a lookup table is used repeatedly to sort superwords, first by their rightmost word, second by their second rightmost word, and so on. Our method on using the superword array also depends on a lookup table to locate quickly all superwords with the same leftmost word.

Existing methods on suffix trees (14), for example, as implemented in the programs REPuter (17) and MUMmer3 (18), provide easy access to segments of variable lengths for seeding overlaps. If the suffix tree is annotated appropriately, then it is easy to check the number of occurrences of a segment in the reads. This is even simpler in enhanced suffix arrays (19), which require only half of the space of suffix trees and provide direct access to occurrence counts. A comprehensive implementation of enhanced suffix arrays is provided by the software tool Vmatch at http://www.vmatch.de.

RESULTS

The algorithms for constructing and using *SW* are implemented in a genome assembly program named PCAP.REP. The PCAP.REP program considers reads with unique superwords in construction of an assembly, whereas the PCAP program considers only reads with unique words in construction of an assembly. The PCAP.REP and PCAP programs were compared on a whole genome dataset from *Histoplasma capsulatum* strain G217B. More than 30% of the genome is occupied by repeat elements, some of which are longer than reads. In other words, a significant number of reads in the dataset contain no unique words of length 12.

The dataset consists of 1 178 375 reads along with 562 071 forward–reverse read pairs. A read pair is produced by sequencing both ends of a subclone, where a read is generated from the 5′ end of each strand of the subclone. A sequencing project for the genome is currently finished up to a set of 261 sequences with a total size of 39.3 Mb. The reads in the dataset have a total of 544 Mb bases with a quality value of at least 20, which corresponds to an average coverage depth of 13.6 for the genome of estimated size 40 Mb.

The two programs were run on the dataset with the following values for their superword/word parameters: (PCAP.REP and PCAP) 12, word length; (PCAP.REP) 110, cut-off on the number of occurrences of a unique superword; (PCAP.REP) 10, cut-off on the number of word levels; (PCAP) 500, cut-off on the number of occurrences of a unique word. For each program, 100 jobs were submitted for computation of overlaps, where each job computes overlaps between the whole set of reads and a subset of reads on a separate processor of 2.4 GHz with 2.5 GB of main memory. The 100 jobs of PCAP.REP produced a total of 272 million overlaps, whereas the 100 jobs of PCAP produced a total of 16 million overlaps. Each PCAP.REP job took 132 min with only 2 s spent on construction of a superword array with 11.6 million positions, whereas each PCAP job took about 36 min. The PCAP.REP job was slower than the PCAP job because the PCAP.REP job computed many more overlaps between repetitive reads than the PCAP job.

PCAP.REP used 974 421 (83%) of the 1 178 375 input reads in its assembly, whereas PCAP used 820 902 (70%) of the input reads in its assembly. In addition, 426,807 (76%) of the 562 071 input read pairs are satisfied in the PCAP.REP assembly, whereas 323,824 (58%) of the input read pairs are satisfied in the PCAP assembly. A read pair is satisfied in an assembly if one read in forward orientation comes before the other read in reverse orientation in the assembly and the distance between the reads in the assembly is close to the estimated size of the subclone.

The PCAP.REP and PCAP assemblies were evaluated in contiguity by computing the N50 lengths of contigs and supercontigs. The contigs in each assembly were partitioned into three groups as follows. Group one was formed by selecting the contigs in order of decreasing length such that the total length of contigs in the group is 20 Mb. Group two was formed by selecting the remaining contigs in order of decreasing length such that the total length of contigs in the group is 19 Mb. Group three contained the remaining contigs, which are very short. The total length of the contigs in groups one and two is 39 Mb, which is the total length of the finished sequences. The N50 length of contigs in a group is the maximum length L such that 50% of all nucleotides are in contigs of length at least L from the group. The N50 lengths of contigs in groups one and two are reported in Table 1. The supercontigs in each assembly were similarly partitioned into three groups. The N50 length of supercontigs in a group is similarly defined. The N50 lengths of supercontigs in groups one and two are reported in Table 2. Tables 1 and 2 show that the PCAP.REP assembly is better than the PCAP assembly in contiguity.

Each of the PCAP.REP and PCAP assemblies was assessed in accuracy through comparison with the set of finished sequences. Reciprocally best matches between the assembly and the finished sequences were found with Cross_Match (P. Green, unpublished data), where a match is reciprocally best if for each of the two regions in the match, the match is the highest-scoring match among all matches involving the region. The reciprocally best matches were used to find differences between the assembly and the finished sequences at the base level and at the sequence level, which are defined as follows.

Table 1. The numbers and lengths of contigs in top two groups for each of the assemblies produced by PCAP.REP and PCAP

Group	Program	Number	N50 length (bp)	Maximum length (bp)	Total length (Mb)
One	PCAP.REP	110	188 832	747 209	20
One	PCAP	136	162 560	461 995	20
Two	PCAP.REP	696	47 730	91 678	19
Two	PCAP	8981	1859	64 190	19

Table 2. The numbers and lengths of supercontigs in top two groups for each of the assemblies produced by PCAP.REP and PCAP

Group	Program	Number	N50 length (bp)	Maximum length (bp)	Total length (Mb)
One	PCAP.REP	14	1 365 435	2 314 423	20
One	PCAP	23	978 523	2 076 521	20
Two	PCAP.REP	101	321 449	888 557	19
Two	PCAP	7164	4396	387 574	19

There are three types of difference at the base level: substitution, deletion from a finished sequence and insertion into a finished sequence. The substitution rate in the reciprocally best matches is the number of substitutions in the matches divided by the total length of the matches. The deletion and insertion rates are similarly defined.

Differences between the assembly and the finished sequences at the sequence level are classified into global and local misassemblies. A global misassembly occurs when a supercontig and a finished sequence with a best match contain different regions of length at least 50 kb, where the cut-off of 50 kb was used in detection of global misassemblies in a chimpanzee genome assembly. Local misassemblies are further partitioned into three types: misordering, interruption and missing.

A misordering event occurs when reciprocally best matching contigs and regions between a supercontig and a finished sequence are in different orders, respectively. For example, a misordering event is detected based on reciprocally best matches if ordered contigs A.4, A.5, A.6, A.7 and A.8 in a supercontig A have reciprocally best matches to a finished sequence in the order A.4, A.7, A.6, A.5 and A.8. If contigs A.5, A.6 and A.7 are very short, it is difficult to determine their order in the supercontig.

An interruption of a supercontig occurs when reciprocally best matches indicate that a gap between two adjacent contigs in the supercontig can be filled with a contig in another supercontig. For example, a gap between adjacent contigs A.5 and A.6 in a supercontig A can be filled with a contig B.1 in another supercontig B if the three contigs have reciprocally best matches to a finished sequence in the order A.5, B.1 and A.6.

A contig in a supercontig is missing when its neighbours have reciprocally best matches to a finished sequence, but the contig has no reciprocally best match to the finished sequence. For example, for ordered contigs A.4, A.5, A.6, A.7 and A.8 in a supercontig A, contigs A.4, A.5, A.7 and A.8 have reciprocally best matches to a finished sequence in this order, but contig A.6 has no reciprocally best match to the finished sequence.

Three difference rates at the base level between the set of finished sequences and each of the PCAP.REP and PCAP assemblies are reported in Table 3. No global misassembly was detected in any of the two assemblies. Three local misassembly rates are reported in Table 4. Tables 3 and 4 show that the PCAP.REP assembly is more accurate than the PCAP assembly.

Table 3. Three difference rates between the set of finished sequences and the set of contig consensus sequences for each of the assemblies produced by PCAP.REP and PCAP

Program	Substitution rate	Deletion rate	Insertion rate
PCAP.REP	0.001376	0.000089	0.000040
PCAP	0.002194	0.000164	0.000136

Table 4. Numbers of local misassembly events per Mb in three categories for each of the assemblies produced by PCAP.REP and PCAP

Program	Misordering	Interruption	Missing
PCAP.REP	0.536441	0.387430	0.298023
PCAP	0.947530	13.265424	0.411969

DISCUSSION

Many existing genome assembly and comparison programs quickly find pairs of sequences with a potential similarity by locating sequences with exact or approximate word matches (1–8,13,20–24). Word matches between sequences are commonly found by using a lookup table and a hashing technique. The lookup table is easy to code and allows approximate word matches. A few existing programs use a suffix tree (17,18,25,26) or enhanced suffix array (http://www.vmatch.de) to locate quickly pairs of sequences with maximal exact segment matches. Methods based on the suffix tree and array use long segment matches to select highly similar pairs of repetitive sequences. The superword array possesses some desirable features of the lookup table and the suffix tree/array. The superword array is easy to code and can be extended to allow approximate superword matches. The number of words in a superword is determined based on the frequency of the superword in the set of sequences.

We have presented an application of a superword array in genome assembly, where the superword array is used to find overlaps between reads with unique superwords. Experimental results produced by PCAP.REP and PCAP on a whole-genome dataset show that a more accurate and contiguous genome assembly was produced by using unique superwords instead of unique words. The construction of the superword array by PCAP.REP was very fast. On the other hand, PCAP.REP spent more time on computation of overlaps between repetitive reads with unique superwords, whereas PCAP spent no time on any overlap between repetitive reads. The cut-off on the number of occurrences of a unique superword was set to a large value because the maximum coverage depth of the genome might be high. A large value for this parameter causes the PCAP.REP job to consider a large number of superword occurrences.

We expect that the superword array will also be useful for locating quickly pairs of similar sequences in genome comparisons. Fast comparison methods based on the superword array would be able to find more orthologous matches in repetitive regions than those based on the lookup table at a slight increase in computation time. Note that the cut-off on the number of occurrences of a unique superword can be set to a value much lower than that used in genome assemblies.

SUPPLEMENTARY DATA

Supplementary Data are available at NAR Online.

ACKNOWLEDGEMENTS

The authors are very grateful to Stefan Kurtz for kindly providing the literature on related previous methods. The authors thank the reviewers for suggestions that improved the presentation of the paper. The authors are supported in part by NIH grant U54 HG003079. Funding to pay the Open Access publication charges for this article was provided by Iowa State University.

Conflict of interest statement. None declared.

REFERENCES

1. Myers,E.W., Sutton,G.G., Delcher,A.L., Dew,I.M., Fasulo,D.P. *et al.* (2000) A whole-genome assembly of Drosophila. *Science*, **287**, 2196–2204.
2. Huson,D.H. *et al.* (2001) Design of a compartmentalized shotgun assembler for the human genome. *Bioinformatics*, **17**, S132–S139.
3. Aparicio,S., Chapman,J., Stupka,E., Putnam,N., Chia,J.M. *et al.* (2002) Whole-genome shotgun assembly and analysis of the genome of Fugu rubripes. *Science*, **297**, 1301–1310.
4. Batzoglou,S., Jaffe,D., Stanley,K., Butler,J., Gnerre,S., Mauceli,E., Berger,B., Mesirov,J.P. and Lander,E.S. (2002) ARACHNE: a whole-genome shotgun assembler. *Genome Res.*, **12**, 177–189.
5. Huang,X., Wang,J., Aluru,S., Yang,S.-P. and Hillier,L. (2003) PCAP: a whole-genome assembly program,. *Genome Res.*, **13**, 2164–2170.
6. Jaffe,D.B., Butler,J., Gnerre,S., Mauceli,E., Lindblad-Toh,K., Mesirov,J.P., Zody,M.C. and Lander,E.S. (2003) Whole-genome sequence assembly for mammalian genomes: ARACHNE 2. *Genome Res.*, **13**, 91–96.
7. Mullikin,J.C. and Ning,Z. (2003) The Phusion assembler. *Genome Res.*, **13**, 81–90.
8. Havlak,P., Chen,R., Durbin,K.J., Egan,A., Ren,Y., Song,X.-Z., Weinstock,G.M. and Gibbs,R. (2004) The Atlas genome assembly system. *Genome Res.*, **14**, 721–732.
9. Dumas,J.P. and Ninio,J. (1982) Efficient algorithms for folding and comparing nucleic acid sequences. *Nucleic Acids Res.*, **10**, 197–206.
10. Pearson,W.R. and Lipman,D. (1988) Improved tools for biological sequence comparison. *Proc. Natl Acad. Sci. USA*, **85**, 2444–2448.
11. Leung,M.Y., Blaisdell,B.E., Burge,C. and Karlin,S. (1991) An efficient algorithm for identifying matches with errors in multiple long molecular sequences. *J. Mol. Biol.*, **221**, 1367–1378.
12. Manber,U. and Myers,E.W. (1993) Suffix arrays: a new method for on-line string searches. *SIAM J. Comput.*, **22**, 935–948.
13. Ma,B., Tromp,J. and Li,M. (2002) PatternHunter: faster and more sensitive homology search. *Bioinformatics*, **18**, 440–445.
14. Gusfield,D. (1997) *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, NY.
15. Manzini,G. and Ferragina,P. (2004) Engineering a lightweight suffix array construction algorithm. *Algorithmica*, **40**, 33–50.
16. Huang,X. (2002) Bio-sequence comparison and applications. In Jiang,T., Xu,Y. and Zhang,M. (eds), *Current Topics in Computational Molecular Biology*. MIT Press, Cambridge, pp. 45–69.
17. Kurtz,S., Choudhuri,J.V., Ohlebusch,E., Schleiermacher,C., Stoye,J. and Giegerich,R. (2001) REPuter: the manifold applications of repeat analysis on a genomic scale. *Nucleic Acids Res.*, **29**, 4633–4642.
18. Kurtz,S., Phillippy,A., Delcher,A.L., Smoot,M., Shumway,M., Antonescu,C. and Salzberg,S.L. (2004) Versatile and open software for comparing large genomes. *Genome Biol.*, **5**, R12.1–R12.9.
19. Abouelhoda,M.I., Kurtz,S. and Ohlebusch,E. (2004) Replacing suffix trees with enhanced suffix arrays. *J. Disc. Alg.*, **2**, 53–86.
20. Altschul,S.F., Gish,W., Miller,W., Myers,E.W. and Lipman,D.J. (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.
21. Ning,Z., Cox,A.J. and Mullikin,J.C. (2001) SSAHA: a fast search method for large DNA databases. *Genome Res.*, **11**, 1725–1729.
22. Kent,W.J. (2002) BLAT: The BLAST-like alignment tool. *Genome Res.*, **12**, 656–664.
23. Brudno,M., Do,C.B., Cooper,G.M., Kim,M.F., Davydov,E., Green,E.D., Sidow,A. and Batzoglou,S. (2003) LAGAN and Multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA. *Genome Res.*, **13**, 721–731.
24. Schwartz,S., Kent,W.J., Smit,A., Zhang,Z., Baertsch,R., Hardison,R., Haussler,D. and Miller,W. (2003) Human-mouse alignments with BLASTZ. *Genome Res.*, **13**, 103–107.
25. Bray,N., Dubchak,I. and Pachter,L. (2003) AVID: a global alignment program. *Genome Res.*, **13**, 97–102.
26. Kalyanaraman,A., Aluru,A., Kothari,S. and Brendel,V. (2003) Efficient clustering of large EST datasets on parallel computers. *Nucleic Acids Res.*, **31**, 2963–2974.