BIOINFORMATICS ORIGINAL PAPER
Vol. 26 no. 14 2010, pages 1699–1703
doi:10.1093/bioinformatics/btq268

Genome analysis

Advance Access publication May 30, 2010

Gap5—editing the billion fragment sequence assembly

James K. Bonfield* and Andrew Whitwham
Wellcome Trust Sanger Institute, Wellcome Trust Genome Campus, Cambridge, CB10 1SA, UK

Associate Editor: Dmitrij Frishman

ABSTRACT

Motivation: Existing sequence assembly editors struggle with the volumes of data now readily available from the latest generation of DNA sequencing instruments.

Results: We describe the Gap5 software along with the data structures and algorithms used that allow it to be scalable. We demonstrate this with an assembly of 1.1 billion sequence fragments and compare the performance with several other programs. We analyse the memory, CPU, I/O usage and file sizes used by Gap5.

Availability and Implementation: Gap5 is part of the Staden Package and is available under an Open Source licence from http://staden.sourceforge.net. It is implemented in C and Tcl/Tk. Currently it works on Unix systems only.

Contact: jkb@sanger.ac.uk

Supplementary information: Supplementary data are available at *Bioinformatics* online.

Received on March 26, 2010; revised on May 17, 2010; accepted on May 18, 2010

1 INTRODUCTION

With the latest wave of DNA sequencing technologies (Bentley *et al.*, 2008; Margulies *et al.*, 2005; Pandey *et al.*, 2008), the number of individual fragments readily available for both mapping and *de novo* assemblies has grown many fold. This has often been coupled with a shortening of each individual fragment. As a consequence, a full mapping of the entire human genome may conceivably have as many as a billion fragments.

While many applications of new sequencing technologies make use of mapped assemblies, *de novo* sequence assembly is still common. These may contain misassemblies or require further 'finishing' work to resolve gaps (Chain *et al.*, 2009). To progress from the draft standard toward finished sequence, we need tools capable of both viewing and editing our large-scale assemblies.

Traditional algorithms used in earlier sequence assembly viewers and editors such as Gap4 (Bonfield *et al.*, 1995), Consed (Gordon *et al.*, 1998), HawkEye (Schatz *et al.*, 2007) and EagleView (Huang and Marth, 2008) tend to scale poorly with the number of fragments. For example, Gap4's memory and CPU usage typically scale linearly with the number of fragments in the assembly. It became clear that the underlying data structures in these older tools are insufficient for the data volumes that we now routinely see.

Recently several viewers including SAMtools (Li *et al.*, 2009), MapView (Bao *et al.*, 2009), IGV (http://www.broadinstitute.org/igv), Tablet (Milne *et al.*, 2010) and NGSView (Arner *et al.*, 2010) have been released that aim to reduce the algorithmic complexity and memory footprint. However, the solutions typically employed by these programs are only amenable for read-only access, with the exception of NGSView that can perform some minor editing tasks.

In addition to algorithmic efficiency, the large increase in the number of DNA fragments has put a strain on our storage requirements. By using data compression methods, the storage burden can be greatly reduced, with the BAM file format being one such recent example. When coupled with an index, compressed BAM files can be randomly accessed.

We present the Gap5 program: a sequence assembly viewer and editor. This encompasses both base by base editing operations as well as high-level contig rearrangements (complementing, breaking and joining). Being able to change data has a substantial impact on the choice of data structures and file formats, which are described below. We also demonstrate the compression techniques used in Gap5 and compare their effectiveness to existing tools.

2 METHODS AND ALGORITHMS

A fundamental challenge for any assembly viewer or editor is how to identify which sequences are visible within a specific region or range, such as the portion of an assembly currently shown on screen. Without an index this range query requires a linear scan, having $O(N)$ complexity, where N is the number of sequences to search through.

Some newer file formats, including MVF (MapView) and CALF (http://www.phrap.org/phredphrap/calf.pdf), make use of an index on the sequence start coordinates. This works well provided we can place a tight upper bound on the maximum length of any sequence. We can rapidly identify sequences entirely within our range query, but to identify those completely spanning the range we have to search backwards, up to the maximum sequence length base pairs away from our range boundaries. If we wish to mix both short and long sequence fragments together this can become inefficient.

To address this programs such as SAMtools (implementing the SAM and BAM file formats) and the UCSC genome browser (Kent *et al.*, 2002) employ spatial indexing (or multidimensional indexing) techniques, e.g. recursive binning and R-Trees (Guttman, 1984). These mechanisms index on both the start and end positions at the same time meaning that we can rapidly interrogate the index to identify sequences visible within a given range, typically in $O(R*\log(N))$ complexity, where N is the length of the contig and R is the size of the query range. For Gap5, we chose a recursive binning algorithm. A contig has a root bin, which in turn has two child bins, repeating in a recursive manner to form a binary tree of bins until we reach a minimum bin size. Sequences and annotations are then placed in the smallest bin that they entirely fit within (Fig. 1).

For an editor another major problem to resolve is how to move data. Storing the absolute position of a sequence leads to algorithmic inefficiencies. Unfortunately this technique was employed in Gap4's database, CAF (Dear *et al.*, 1998) and ACE file formats, and even in newer short-read formats such as CALF, SAM, BAM and MVF. If we perform an edit that moves sequences within a contig, such as making an insertion or joining the contig to another, then we need to alter the location of potentially every sequence within that contig.

*To whom correspondence should be addressed.

© The Author(s) 2010. Published by Oxford University Press.
This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/2.5), which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

Downloaded from http://bioinformatics.oxfordjournals.org/ at Wellcome Trust Genome Campus on February 1, 2012

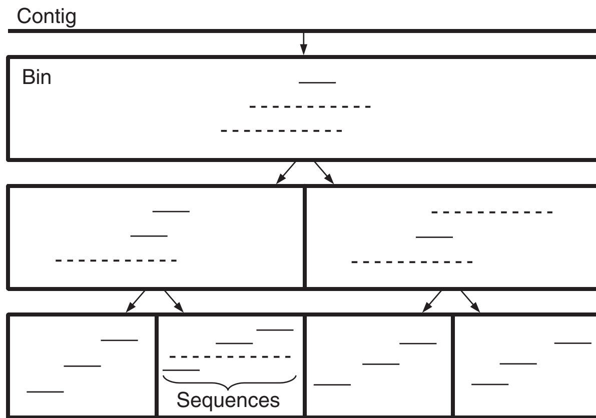

Fig. 1. Binning tree containing sequences from two libraries (represented by solid and dashed lines). Information about the sequence positions and pairings is stored in the bin records, while the sequence names, DNA and qualities are held in the sequence records.

One solution to this comes from making sure that the location of sequences and annotations are stored relative to the bin they have been placed within. Additionally, the location of a bin itself is stored relative to its parent bin. With all positional data being relative to the parent object, we can now shift entire portions of a contig with just $O(\log(N))$ operations.

There is one further editing operation that needs special attention: complementing a contig. If we choose to have a single status flag on the contig indicating whether this entire contig is to be viewed in the original or complemented orientation, we will have a problem when we wish to join it to a contig of the opposite orientation. For example, if we wish to join contig A to the complement of contig B, then the resulting contig will have a mixture of complemented and uncomplemented data. We do not wish to actually reverse complement the data in contig B as this may require millions of changes to be made. To resolve this, each bin has a flag indicating whether it and its children are complemented with respect to its parent bin. This permits multiple complement and join operations to occur with the minimum of data editing.

The bins can also serve as a way to cache data views at different zoom levels. When showing a narrow region at high magnification, we will query deep into our tree; when showing a very large region at low magnification, we may only need to query the top few levels of bins to achieve the desired resolution. So far this has only been implemented experimentally to store sequence depth data. By storing a fixed number of data points per bin, regardless of the number of bases they span, we can rapidly draw the sequence depth at any zoom level with a minimum amount of disk I/O. These bin 'tracks' simply act as caches for the actual algorithms that obtain the data to plot. They are invalidated after some types of edits and are recomputed on demand. In a similar, albeit simpler manner we use one layer in the bin tree to store cached fragments of the consensus. This means that after changing the data we can mark small portions of the consensus as invalid, reducing the overhead of recomputing it.

When interactively scrolling through a contig, most of the data required to perform the range query will have been recently loaded for a previous query, as the path down the bin tree will typically be the same except for a few leaf nodes. By keeping a cache of recently accessed records in memory, we substantially reduce the I/O overhead. To achieve this all database records are accessed via a data structure, we term as a HacheTable: a caching hash table. The programmatic interface to this gives the appearance of all database records being held within memory; however, only the most recently used items are stored with older items being discarded to keep memory usage low. The hache hit rate while scrolling is typically >99%. The same HacheTable is also used to keep track of edited objects by locking these items to prevent

Fig. 2. Contig editor, showing quality values by gray scales and mismatches to the consensus by base color.

them from being discarded. When the user saves changes to disk the lock status is used as an indicator of which objects to save.

3 IMPLEMENTATION

Gap5 is primarily written in C (Kernighan. and Ritchie, 1988) for efficiency and Tcl/Tk (Ousterhout, 1990) for the graphical user interface. The use of Tcl also means we have an inbuilt scripting language allowing user-controlled automation of many of Gap5's capabilities. Many of the Staden Package libraries used by Gap4 have also been reused for Gap5, making Gap5 a direct descendent of Gap4. However, many of the algorithms and data structures first appeared in 'tg_view': a prototype text-based editor and viewer. This was first publicly released in 2007 (unpublished data) as the *tgap* package and was initially used as a viewer for MAQ (Li *et al.*, 2008) alignments.

For visualization, Gap5 shares a lot of common features with its ancestor Gap4. The contig editor (Fig. 2) displays sequences along with their per-base quality values as gray scales. Discrepancies between the sequence and the consensus may be automatically highlighted by either color or symbol. Sequence names and/or mapping qualities, if known, are shown in the left panel. To save vertical space multiple sequences may be packed onto one display line.

To obtain a broader view of an assembly the sequences may be shown pictorially using the template display. This draws one line per sequencing template, with the horizontal size and placement governed by the location of the forward and reverse sequence fragments. Colors are used to distinguish templates where the forward and reverse fragments are in separate contigs, have an inconsistent (unexpected) orientation or are single-ended sequencing templates only. We draw either a traditional assembly illustration with the Y coordinate being used simply to separate overlapping sequences (Supplementary Material), or a LookSeq (Manske and Kwiatkowski, 2009) style plot, where the Y coordinate is governed by the insert size. This latter type of plot is particularly effective at identifying regions where indels have occurred (Fig. 3).

We observe that to draw the template display, we need to know only sequence start and end coordinates, mapping scores, mate pairs and a few status flags. The sequence structures constitute the vast bulk of the database size so it is costly to extract this information from the sequence structures themselves. However, as previously described, a contig bin contains references to sequences and annotations along with their locations. To optimize the template display, we also store additional information (mapping score, mate

Downloaded from http://bioinformatics.oxfordjournals.org/ at Wellcome Trust Genome Campus on February 1, 2012

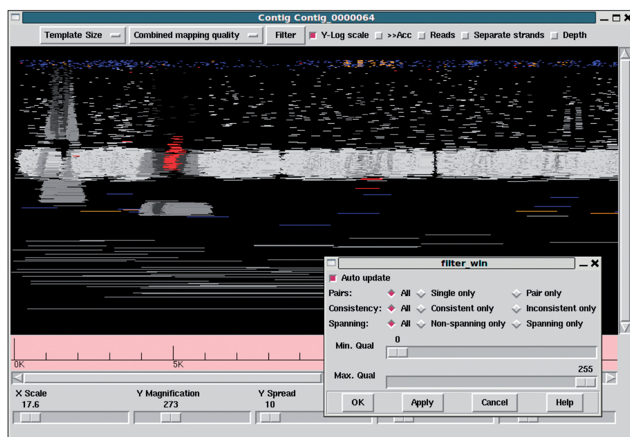

Fig. 3. Template display showing a mapped assembly with a short insert Illumina library and a long insert capillary library. The Y-axis here shows insert size, while the X-axis is the position within the contig. A genomic insertion is visible at around 5 kb, identified by the jump in average insert size for the Illumina library. Also visible is the filter subwindow. The template colors used are red: inconsistent read-pair orientation; blue: single-ended template; orange: template spanning two contigs; otherwise gray-scale: the mapping quality of the DNA fragments.

pair record and status flags) in the bin. This dramatically reduces the amount of disk I/O required to draw the template display. It also means that if we only desire to view graphical summaries, we can create an assembly database with no sequence names, DNA or quality values.

In order to achieve editing capabilities, Gap5 does not use a flat-file format as objects can and do change size—even base substitutions may change the size of the sequence we store once it has been compressed. We reuse the lightweight Gap4 database engine, although the substantial schema differences mean that the two programs are incompatible. Tools are provided to migrate from Gap4 to Gap5 via use of the caftools package and the *caf2baf* perl script. (Note though that at the time of writing this article the total functionality of Gap4 still greatly exceeds the functionality of Gap5.)

The construction of a Gap5 database is achieved using the separate *tg_index* tool. This supports reading from ACE, SAM, BAM and our own local BAF format. It can be a time-consuming operation on large datasets, converting in the order of 40 000 sequences per second, although this is only around 70% slower than converting SAM to BAM using SAMtools. Tg_index is also currently the most memory hungry part of the package, using 10 Gb on a 1.1 billion read test set. Gap5 may be used to output all or a portion of an assembly in ACE, SAM, BAM, BAF, CAF, fasta or fastq format.

In order to keep the database compact, by default Gap5 uses the Zlib (Deutsch and Gailly, 1996) compression layer. This was found to be a good compromise between space and efficiency. For cases where space is the primary constraint, Gap5 can use LZMA2 instead (implemented using the XZ-utils package: http://tukaani.org/xz). It was discovered that attempting to individually compress each sequence separately has a high overhead, so prior to compression Gap5 collates up to 1024 sequences and annotations together. To further improve compression ratios, within each of these blocks we reorder the data by content type. For example, a block of 1024 sequences will yield 1024 names, 1024 DNA strings and 1024 quality strings. Each type of data is then compressed independently, ensuring that the Huffman tables used by Zlib are optimally tuned to each data type. Where applicable, numerical records such as the sorted position data within bin range arrays are differentiated to store successive deltas. All numerical values are stored to variable size depending on the absolute magnitude of the value. The combined impact of these methods are considerable on compression ratios and the primary reason for Gap5 databases being considerably smaller than BAM files.

Table 1. Efficiency of opening and viewing an assembly

Program	Dataset	CPU (s)	Memory (MiB)	File size
Gap4	A	149	6784	4 823 620 112
Consed	A	363	6270	3 838 652 583
EagleView	A	385	10 044	2 461 728 347
NGSView	A	0.2	36	4 197 720 064
MapView[a]	A	3.0	32	558 031 038
IGV	A	5.2	118	186 611 223
SAMtools	A	1.2	34	186 611 223
Gap5	A	0.2	15	139 030 256
IGV	B	5.0	110	43 832 012 709
SAMtools	B	1.1	49	43 832 012 709
Gap5	B	0.4	22	32 153 736 504

[a]Tested on a 32-bit linux system due to lack of a Mono environment on the main 64-bit test system.
'MiB' is 1 048 576 bytes—a mebibyte. Dataset A is 6.6 million 44 bp reads mapped to a single 44 Mb contig. Dataset B is 1.1 billion reads (mostly 36 bp) mapped to all human chromosomes. Program versions: EagleView 2.2, Gap5 1.2.7, SAMtools 0.1.7a, MapView 3.4.1, Consed 19.0, Gap4 4.11 and IGV 1.4.

4 RESULTS

For an initial test, we chose to use the data presented in the MapView paper: 6.6 million 44 bp reads aligned in a single 44 Mb contig. We converted this file to a variety of formats taking care to include the appropriate data (including sequence names, bases and quality) supported by all formats and no more. We then measured the CPU time taken to start up the program, open the assembly and view sequence assembly at the start of the first contig. Table 1 presents these results as dataset A, along with the programs native file sizes. See the Supplementary Material for a more complete break down on the assembly file sizes.

As can be seen, the programs mostly cluster into two groups, with EagleView, Gap4 and Consed being very demanding on both memory and CPU. These three also had the largest disk space requirements. The last four—MapView, IGV, SAMtools and Gap5—all demonstrate acceptably low resource requirements for both CPU and memory, while also using substantially less disk space. NGSView is very CPU and memory efficient, but is inefficient on disk space usage. Note that the CPU time and memory also includes the constant overhead of launching the programs, so it may not accurately reflect the relative positions of the last five programs when faced with much larger datasets.

To further test scalability we used a 1000 genomes (http://www.1000genomes.org/) project SAM file containing 1.1 billion reads from the NA19240 sample. Note that this BAM file contained only mapped data with the only auxiliary records being the read

Downloaded from http://bioinformatics.oxfordjournals.org/ at Wellcome Trust Genome Campus on February 1, 2012

Table 2. I/O efficiency on dataset A

Program	Operation	I/O calls	Bytes r/w (KiB)
gap4	Open + view	138 928 263	3 418 452
gap5	Open + view	81	116
samtools	Open + view	9	140
gap4	Move to 20 Mb	312	4
gap5	Move to 20 Mb	58	101
samtools	Move to 20 Mb	33	221
gap4	Scroll to 21 Mb	310 266	6288
gap5	Scroll to 21 Mb	476	4616
samtools	Scroll to 21 Mb	10 850	47 050
gap4	Break contig	31 208 502	689 953
gap5	Break contig	1794	823
gap4	Join contig	79 387 624	1 653 908
gap5	Join contig	187	2

I/O operations showing the number of I/O calls (lseek, read, write, pread, pwrite) for opening the database and displaying the first contig, moving to position 20 Mb in the contig, scrolling to 21 Mb in 1 kb increments, for breaking the contig in two at 20 Mb and joining it together again. For a more complete break down of the I/O calls used see the Supplementary Material.

group as storing this additional information is still experimental in Gap5. The results for these are listed as dataset B in Table 1. It is evident from this that the scalability problems have largely been solved by several tools including Gap5.

As Gap5 uses a simple database rather than a flat file, I/O efficiency could be a concern. So to test I/O efficiency we compared Gap5 with Gap4 and SAMtools tview on the 6.6 million read dataset A. The results in Table 2 demonstrate that the start up cost of Gap5 is low as it does not load the entire index into memory, but a consequence of using a database means that we require many more disk seeks than SAMtools. Gap4 in comparison is very I/O intensive as it loads partial information about every sequence when it opens the database.

When scrolling along a contig view both Gap4 and Gap5 demonstrate a minimal amount of additional data loaded due to on-the-fly caching in Gap5 and having preloaded most of the data in Gap4. It is clear that Gap5's approach of blocking 1024 sequences together per database record dramatically reduces the number of I/O calls. SAMtools demonstrates an apparent lack of data caching in this test, but was still fast and responsive.

The complexity of editing operations is where Gap5 really stands out against Gap4. The inability to reposition large numbers of sequences without individually editing each one causes Gap4 to generate millions of I/O calls when breaking contigs in two or joining them together.

To verify the efficiency of Gap5 against a 1 billion read assembly, we repeated these tests on dataset B. As can be seen in Table 3, edits still require a relatively small amount of I/O. The speed was also acceptable: to perform all 10 breaks, joins, substitutions and insertions took 12 s of CPU time. We could not compare editing of this dataset against Gap4 due to time and memory constraints, but for viewing purposes we also tested SAMtools. In contrast with the smaller set, we observe that SAMtools reads far more data when

Table 3. I/O efficiency on data set B

Program	Operation	I/O calls	Bytes r/w (KiB)
gap5	Open + view	339	774
samtools	Open + view	146	8516
gap5	Move to 100 Mb	76	179
samtools	Move to 100 Mb	15	138
gap5	Scroll to 101 Mb	645	10 373
samtools	Scroll to 101 Mb	12 192	81 560
gap5	Break contig	2859	805
gap5	Join contig	228	135
gap5	Substitution	145	52
gap5	Insertion	1047	104

I/O operations showing the number of I/O calls (lseek, read, write, pread, pwrite) with dataset B. The contig viewed was Chromosome 1. Breaking and joining contig measurements were averaged over 10 contigs, for Chr4 to Chr13. The substitution and insertion tests were averaged from single base edits at 10 locations spread over ChrX.

Table 4. Data compression of data set A

File format	Compression tool	File size
sam	–	885 524 410
bam	(bgzf)	186 486 871
sam	gzip	179 625 250
sam	7zip	144 426 218
gap5	(zlib)	137 783 736
gap5	(lzma2)	115 331 272
sam	paq8o9	86 875 700

Compression tools listed in parentheses denote algorithms internal to either SAMtools or Gap5. All others are external command-line tools.

opening the assembly. This is due to completely loading the BAM index file into memory. The lack of caching in SAMtools is again evident during the scrolling test.

To evaluate storage size, we experimented with a variety of compression algorithms on the sam files exported from Gap5 and compared these with Gap5's native format, using both the lz77 (zlib) and lzma2 (xz-utils) algorithms. For speed reasons, we only tested this with the smaller dataset A. Table 4 presents these findings. The PAQ algorithm (Mahoney, 2005) and variants have won the Hutter prize for compression multiple times and can be considered as at the cutting edge for general purpose compression, regardless of the cost in CPU. While not practical—it took 26 h to compress the sam file—it is a useful baseline to compare ourselves against. For comparison, *tg_index* produced the Gap5 database in 144 s when using lz77 and 502 s using lzma2. It is clear that Gap5 has not had to compromise greatly on storage space in order to achieve both random access and editability of data.

Tg_index has the ability to ignore certain types of data or to replace them with blank data, such as producing minimal names, setting all quality values to zero, or even replacing all base calls

Downloaded from http://bioinformatics.oxfordjournals.org/ at Wellcome Trust Genome Campus on February 1, 2012

Table 5. File size by content type, data set A

Data type	File size (%)	Bits per seq.	Bits per base
bin/range	4.7	7.75	0.18
Seq bases	23.5	38.83	0.88
Seq quality	42.6	70.36	1.60
Seq name	25.6	42.28	0.96
Seq other	3.7	6.08	0.14

File sizes from tg_index -z 16384 -d *data_type*. 'Seq other' here is a general per-sequence overhead. The 'bin/range' type includes everything needed to draw the Template Display window; sequence positions, mapping quality and read pairings.

with N. From this, an analysis of the storage per type of data is presented in Table 5.

It is clear that the quality values constitutes the bulk of the file size, with the DNA sequence taking up less than 1 bit per base call. This figure is substantially less than the expected 2 bits per base due to redundancy in the sequence depth (7×) and so clearly the results will differ when tested on other datasets.

5 DISCUSSION

We have demonstrated that we can keep and sometimes improve upon the CPU, memory and I/O efficiency of the next-generation assembly viewers, while also supporting editing capabilities. This is a marked improvement over the Gap4 program. However, it is clear that performance is just one aspect and utility also needs to be considered. Currently, Gap4 offers a much richer set of tools than Gap5 and is also available on a broader range of platforms. Over time, we expect to duplicate the most important Gap4 features in Gap5 and also plan to port Gap5 to Microsoft Windows.

There are still some performance issues even with Gap5 as intrinsically certain algorithms will not be possible to get below $O(N)$ complexity, such as plotting an entire chromosome or identifying all local alignments in an entire genome. Some algorithms can benefit from precomputation of results at a cost of increased storage, which so far we have only implemented for consensus caching. We have outlined ways that the binning tree can be used to store additional precomputed depth data. This aspect of Gap5 is still largely unexplored, but we envisage a variety of additional cached tracks for rapid visualization in the template display. Further analysis of the I/O patterns reveals that the bulk of I/O calls while breaking contigs are manipulating the bin tree. SAMtools and the UCSC Genome Browser both use trees with eight children per node, rather than the binary tree implemented in Gap5. Implementing a similar change to Gap5 should further improve I/O performance.

It is likely that users will want to keep both their standard alignment format data, such as BAM files, as well as using Gap5 for viewing and possibly editing. The fact that Gap5 is efficient in space helps, but it is clear that this is an additional cost over and above the storage requirements for the input data. One possible solution to this is to observe that indexing just the sequence positions (*tg_index -d blank*) is only an extra 5% on top of the BAM format. It may be possible to get Gap5 to extract names, sequences and qualities from BAM while still retaining the positional index for use in the template display. The next logical step is to implement a copy-on-write scheme where only edited sequences get added to the Gap5 database. This will bring the additional overheads of editing to an acceptable level.

ACKNOWLEDGEMENTS

Many thanks to Rodger Staden and his past team at the Medical Research Council Laboratory of Molecular Biology, whose work greatly simplified the development of Gap5. We acknowledge Robert Davies and the other members of his group at the Wellcome Trust Sanger Institute for numerous discussions during this work, Li Heng for providing the code for reading maq and sam file formats and the 1000 Genomes project for providing us with data.

Funding: Wellcome Trust (grant number 077200/Z/05/Z); the Medical Research Council.

Conflict of Interest: none declared.

REFERENCES

Arner,E. *et al.* (2010) NGSView: an extensible open source editor for next-generation sequencing data. *Bioinformatics*, **26**, 125–126.

Bao,H. *et al.* (2009) MapView: visualization of short reads alignment on a desktop computer. *Bioinformatics*, **25**, 1554–1555.

Bentley,D.R. *et al.* (2008) Accurate whole human genome sequencing using reversible terminator chemistry. *Nature*, **456**, 53–59.

Bonfield,J.K. *et al.* (1995) A new DNA sequence assembly program. *Nucleic Acids Res.*, **23**, 4992–4999.

Chain,P.S.G. *et al.* (2009) Genome project standards in a new era of sequencing. *Science*, **326**, 236–237.

Dear,S. *et al.* (1998) Sequence assembly with CAFTOOLS. *Genome Res.*, **8**, 260–267.

Deutsch,P. and Gailly,J.L. (1996) Zlib compressed data format specification version 3.3. RFC 1950. Available at http://www.ietf.org/rfc/rfc1950.txt (last accessed date June 2, 2010).

Gordon,D. *et al.* (1998) Consed: a graphical tool for sequence finishing. *Genome Res.*, **8**, 195–202.

Guttman,A. (1984) R-Trees: a dynamic index structure for spatial searching. In Yormark,B. (ed.) *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*. ACM Press, pp. 47–57.

Huang,W. and Marth,G. (2008) EagleView: a genome assembly viewer for next-generation sequencing technologies. *Genome Res.*, **18**, 1538–1543.

Kent,W. J. *et al.* (2002) The human genome browser at UCSC. *Genome Res.*, **12**, 996–1006.

Kernighan,B.W. and Ritchie,D.M. (1988) *The C Programming Language*. Prentice Hall, Upper Saddle River, New Jersey.

Li,H. *et al.* (2008) Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Res.*, **18**, 1851–1858.

Li,H. *et al.* (2009) The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, **16**, 2078–2079.

Mahoney,M.V. (2005) Adaptive weighing of context models for lossless data compression. *Technical Report CS-2005-16*, Florida Institute of Technology.

Manske,M. and Kwiatkowski,D.P. (2009) LookSeq: a browser-based viewer for deep sequencing data. *Genome Res.*, **19**, 2125–2132.

Margulies,M. *et al.* (2005) Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, **437**, 376–380.

Milne,I. *et al.* (2010). Tablet - next generation sequence assembly visualization. *Bioinformatics*, **26**, 401–402.

Ousterhout,J.K. (1990) Tcl: An embeddable command language. In *Proceedings USENIX Winter Conference*, USENIX Association, Berkeley, CA, pp. 133–146.

Pandey,V. *et al.* (2008) Applied biosystems SOLiD system: ligation-based sequencing. In *Next-Generation Genome Sequencing*. Wiley-VCH, Berlin, Germany, pp. 29–41.

Schatz, M. C. *et al.* (2007) Hawkeye: an interactive visual analytics tool for genome assemblies. *Genome Biol.*, **8**, R34.

Downloaded from http://bioinformatics.oxfordjournals.org/ at Wellcome Trust Genome Campus on February 1, 2012