BIOINFORMATICS ORIGINAL PAPER Vol. 21 no. 8 2005, pages 1371–1375
doi:10.1093/bioinformatics/bti184

Genome analysis

A graph based algorithm for generating EST consensus sequences

Ketil Malde[1,*], Eivind Coward[1] and Inge Jonassen[2]

[1]Department of Informatics, University of Bergen and [2]Computational Biology Unit, Bergen Centre for Computational Sciences and Department of Informatics, University of Bergen, Norway

Received on June 28, 2004; revised on September 15, 2004; accepted on November 24, 2004
Advance Access publication November 30, 2004

ABSTRACT

Motivation: EST sequences constitute an abundant, yet error prone resource for computational biology. Expressed sequences are important in gene discovery and identification, and they are also crucial for the discovery and classification of alternative splicing. An important challenge when processing EST sequences is the reconstruction of mRNA by assembling EST clusters into consensus sequences.

Results: In contrast to the more established assembly tools, we propose an algorithm that constructs a graph over sequence fragments of fixed size, and produces consensus sequences as traversals of this graph. We provide a tool implementing this algorithm, and perform an experiment where the consensus sequences produced by our implementation, as well as by currently available tools, are compared to mRNA. The results show that our proposed algorithm in a majority of the cases produces consensus of higher quality than the established sequence assemblers and at a competitive speed.

Availability: The source code for the implementation is available under a GPL license from http://www.ii.uib.no/~ketil/ bioinformatics/

Contact: ketil@ii.uib.no

INTRODUCTION

ESTs are produced by one-shot sequencing of cDNAs produced by cloning mRNA. The sequences are easy to produce, and are a useful and versatile resource of gene sequence data. However, they are prone to sequencing errors and vector sequence contamination, and methods for EST analysis need to take this into account.

In order to reconstruct the original mRNA, EST analysis normally entails a clustering stage—attempting to group ESTs according to originating gene, followed by an assembly or consensus stage—aiming to determine for each cluster one or more consensus sequences to which the sequences can be globally aligned. Ideally, one consensus sequence is produced for every mRNA isoform of the gene, but in practice, low coverage regions and high error rates make this goal difficult to attain.

Recent works (Mironov *et al*., 1999; Modrek and Lee, 2001) indicate that alternative transcripts are very common, and consequently, we must expect ESTs in a cluster to contain fragments, not only from a single correct mRNA sequence, but from several equally correct, related sequences.

*To whom correspondence should be addressed.

The presence of multiple correct assemblies contributes to making the assembly of EST data inherently different from assembling genomic DNA.

In addition, EST data are generally considered to be more prone to sequencing errors. However, compared to genome assembly, repeats are less of a problem for assembling an individual gene, since the coding sequence of a gene is unlikely to contain repeats. (It is still an important consideration for EST *clustering*, of course.) The EST assembly problem is thus different enough from genomic assembly to warrant specialized tools (Pertea *et al*., 2003).

The classical approaches to sequence assembly use the 'overlap–layout–consensus' approach; that is, they use pairwise sequence alignments to find a best fit, and build contigs by merging sequences that overlap.

In the ideal case, we can construct a directed graph where each sequence is a vertex, and edges represent overlaps between sequences; i.e. there is an edge from s_1 to s_2 if and only if a suffix of s_1 matches a prefix of s_2. A correct assembly of the set of sequences is then constructed by finding a Hamiltonian path through this graph.

There exist many sequence assemblers that can be used for constructing consensus sequences of ESTs. Some of the more popular are Phrap (Green, 1996), the TIGR Assembler (Sutton *et al*., 1995) and CAP3 (Huang and Madan, 1999). Liang *et al*. (2000) provide a thorough comparison of these tools for the EST clustering problem. However, common to all of these is that they are primarily designed for assembling genomic sequences, rather than ESTs, and recent efforts to improve the state of the art in sequence assembly tend to specialize even more for the genomic assembly problem (Batzoglou *et al*., 2002; Jaffe *et al*., 2003; Myers *et al*., 2000).

A different approach to sequence assembly is based on finding Eulerian paths in a graph. Based on work on 'sequencing by hybridization', the idea is to break down sequencing data into fixed length, overlapping fragments, or k-tuples (Idury and Waterman, 1995). It is then possible to extract a consensus sequence as a path through a graph over these fragments. However, this graph is complicated by read errors, and effective algorithms to eliminate these errors are essential (Pevzner *et al*., 2001).

The advantage of this method is partly efficiency, since it computes the assembly by finding an Eulerian path through a graph, which is computable in linear time, while computing the assembly from the overlap graph is NP-complete (the Hamiltonian path problem).

Since the ESTs in a cluster corresponding to a gene originate from multiple but related mRNA sequences, the graph resulting from applying this approach will partly be determined by the splice

structure of the gene, and is consequently sometimes referred to as a *splice graph* (Heber *et al*., 2002).

A splice graph is thus in itself a very descriptive way to present a cluster of ESTs, and can be more revealing of the underlying splice structure than assembled consensus sequences. However, as most tools deal with sequences rather than graphs, it is for many purposes still preferable to represent clusters as consensus sequences.

In the following we present an alternative algorithm for constructing EST consensus sequences. Similarly to the other graph based approaches, we construct a graph over sequence fragments representing the data set. However, we use a different algorithm that does not explicitly try to construct an Eulerian path, nor calculate sequence alignments or overlaps. Instead we traverse the graph greedily, trying to pursue a path through the graph so that each branch followed is as consistent with the previous one as possible.

A complete EST analysis process incorporates many stages, from base calling, quality and vector clipping, and repeat masking, through sequence clustering and assembly, to analysis and detection of transcriptional features like SNPs and splice variants (Staden, 1996; Chevreux *et al*., 2004). We therefore wish to emphasize that the present algorithm and tool only deals with the sequence *assembly* stage, and that other tools must be used for the other parts of the EST analysis tool chain.

BACKGROUND

We define a *word graph* for a data set $S \subset \Sigma^*$ and with word length k as a directed graph (V, E) where the set of nodes V is the set of all $(k-1)$-words in S, the edges $E = \{(xw, wy) | x, y \in \Sigma, xwy \text{ is a } k\text{-word in } S\}$.

For our purposes, S is a set of sequences over the alphabet $\Sigma = \{A, C, G, T\}$, and then nodes then represent all length $(k-1)$-substrings, while edges represent the length k substrings of the sequences in S. We also define the *weight* of an edge as the number of sequences in S containing the word represented by the edge. [The word graph is a subgraph of the *de Bruijn graph*; see, e.g. Skiena and Sundaram (1993) for a more elaborate description.]

Different features of the data set will give rise to corresponding properties of the word graph. For example, a single read error will cause the word graph to branch into paths that merge again after k nodes. Since the chance for the same random error occurring in the same position in two sequences is very low, we expect one of the paths to have a weight of one, and given good coverage, the path representing the correct sequence to have a higher weight. For non-random polymorphisms, it is more likely that the different paths will be supported by several sequences. In addition, if any k-word occurs more than once in a sequence, it will lead to a cycle in the graph.

Word graphs for real data will be complicated by the combination of these features, and the 'raw' graph often becomes a complex tangle of edges, something that moderated the success of the early Eulerian graph sequence assemblers (Pevzner *et al*., 2001).

The present algorithm does not try to construct an Eulerian graph, but instead takes into account the weight of edges to eliminate read errors.

ALGORITHM

Based on the word graph for a set of sequences, we construct consensus sequences by finding *greedy paths* through the graph. The graph is traversed, trying to follow edges so that the path is consistent with at least a subset of the sequences.

The algorithm takes as a parameter the minimum weight threshold, which indicates the number of sequences that need to support a particular edge in the graph in order for us to consider it reliable—or in other words, if the number of sequences supporting an edge is lower than this threshold, we suspect it is due to sequence errors, rather than actual sequence differences. This threshold is used as a stop criterion; when no edges with weights exceeding the threshold remain, the algorithm terminates.

Given a word length k, the algorithm starts out by constructing the corresponding k-word graph. The graph is then examined to identify the heaviest edge—i.e. the k-word occurring in most sequences—and the set of sequences supporting it.

We can now construct the *greedy path* in each direction by traversing the graph until we reach a junction. For each branch out of the junction, we examine the set of sequences supporting it. We select the branch supporting most of the sequences that also support the heaviest edge.

At the next junction, we again examine the set of sequences supporting each branch, and select the one supporting most of the sequences that supported the previously taken branch. This process is repeated, until we reach a node with no outgoing edges.

One problem that arises is the presence of cycles in the graph. If a word is repeated in a consensus sequence, we risk the danger of looping infinitely. This situation is avoided by maintaining a map of the current position in the sequences. When the consensus is extended, only sequences where the extension would increase the current position are taken into account. This allows the consensus sequence to contain multiple occurrences of the same word, but only if it can be represented by different positions in the sequence data. The map is also used in branch selection, gradually increasing the weight of sequences that align well to the consensus. If a word occurs multiple times in a sequence, its position cannot be unambigously determined in that sequence. In order to avoid this ambiguitiy, the selection of starting point penalizes the weight of multiply-occurring words.

When the greedy path cannot be extended in either direction, it is returned as a consensus sequence, and the next path can then be constructed, starting from the heaviest edge not in any previous sequence.

We used this algorithm to implement an experimental transcript assembly/consensus tool, *xtract*. In this implementation, the word graph is represented by a finite map (a.k.a. dictionary), mapping k-words to sequence and position pairs. This makes it fast [an $O(\log n)$ operation] to check whether a given k-word exists, and to retrieve the list of occurrences in the data set.

Nodes and edges in the word graph can be extracted from this structure; each k-word entry represents an edge from the $(k-1)$-prefix to the $(k-1)$-suffix of the word; the nodes can be inferred implicitly from the edges.

The implementation was written in Haskell and compiled with the GHC compiler.

RESULTS

Since UniGene (Boguski and Schuler, 1995) contains both cDNA ESTs as well as full-length mRNAs, we decided to test the accuracy of our assembly algorithm on the ESTs and use the mRNA as

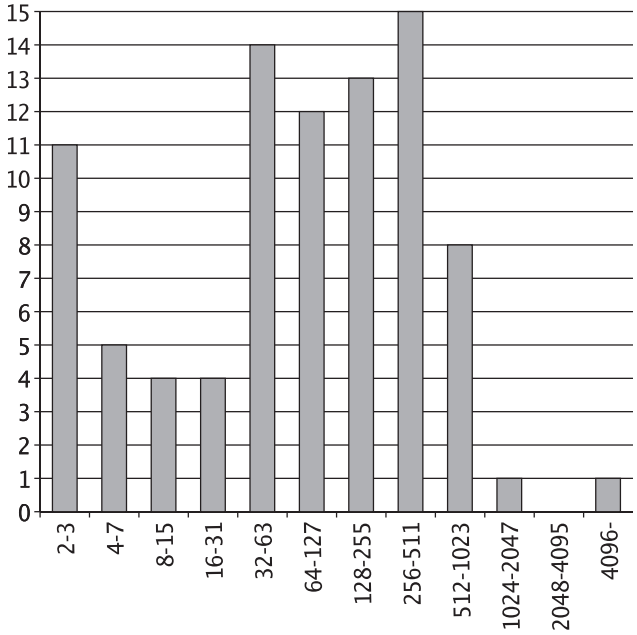

Fig. 1. The distribution of cluster sizes.

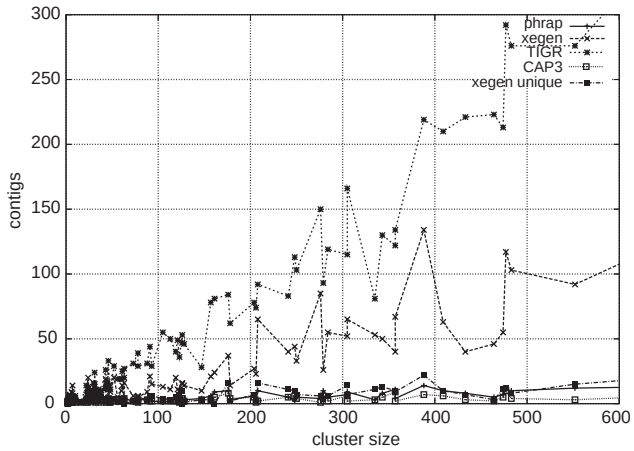

Fig. 2. The number of contigs produced by each sequence assembler plotted against cluster sizes. The line labeled *unique* is the number of sequences after redundancy has been eliminated with *cd-hit* (see Discussion). The largest clusters are not shown in the diagram due to space constraints.

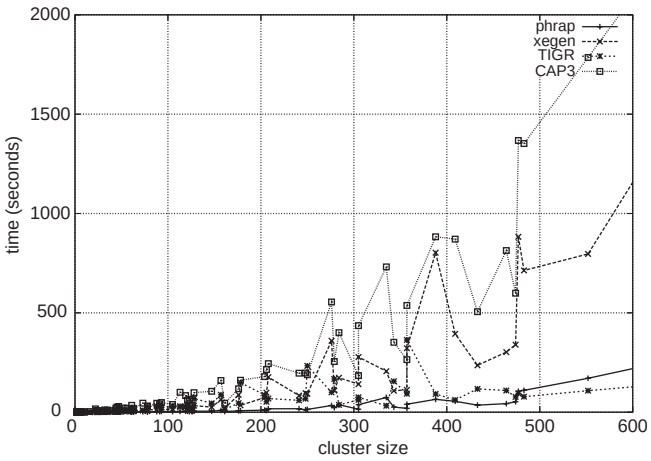

Fig. 3. Running times for each sequence assembler plotted against cluster sizes. The largest clusters are not shown due to space constraints.

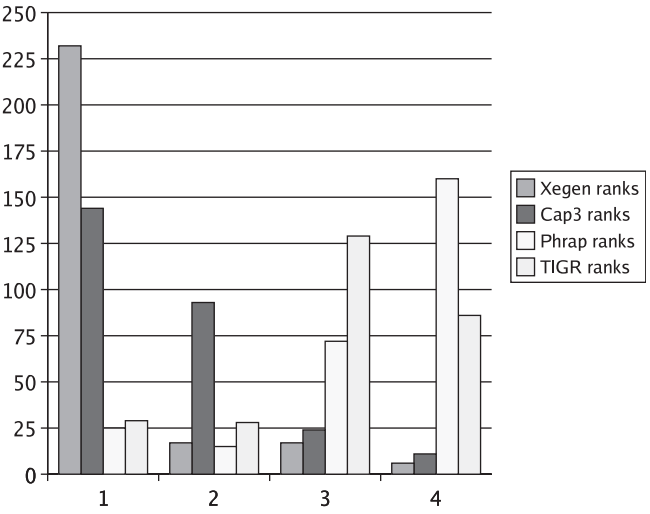

Fig. 4. The ranks of best contig matching each mRNA for each of the sequence assemblers.

reference. In most cases, sequences are annotated as either mRNA or cDNA, sequences with neither designation being treated as cDNA.

We extracted the 20 427 sequences contained in the first 100 UniGene clusters (Boguski and Schuler, 1995), and masked them using RepeatMasker. Since UniGene uses annotation in addition to sequence information when producing the clustering, the sequences were reclustered based on sequence information only, using xsact (Malde *et al*., 2003), with a block size of 20 and a match threshold of 75 (i.e. sequences are clustered together if they have exactly matching blocks of length 20 or more, whose sum of lengths is at least 75).

The full length mRNAs (274 sequences, identified by annotation) were then removed from the clusters, leaving only the EST sequences. The largest cluster was comprised of approximately 7500 sequences and 64 mRNAs. The complete distribution of cluster sizes is shown in Figure 1.

We then ran Phrap, TIGR and CAP3, as well as our algorithm on each of the resulting clusters. For easy comparison, all computations were performed on a Sun Fire 880 computer.

For each cluster, the number of contigs produced by the different programs is presented in Figure 2. The running time of each sequence assembler was measured, and the results are presented in Figure 3.

For each cluster, we compared all the produced contigs against the mRNAs in the cluster. We extracted from the output of each assembler the contig with the highest BLAST bitscore to the mRNA and ranked them. In the case of ties, contigs producing the same score were given the same rank. The ranks were counted for each assembler, and the results are displayed in Figure 4.

The aligment of the consensus sequences for one of the clusters in the data set is shown in Figure 5. Only CAP3 and xtract manage

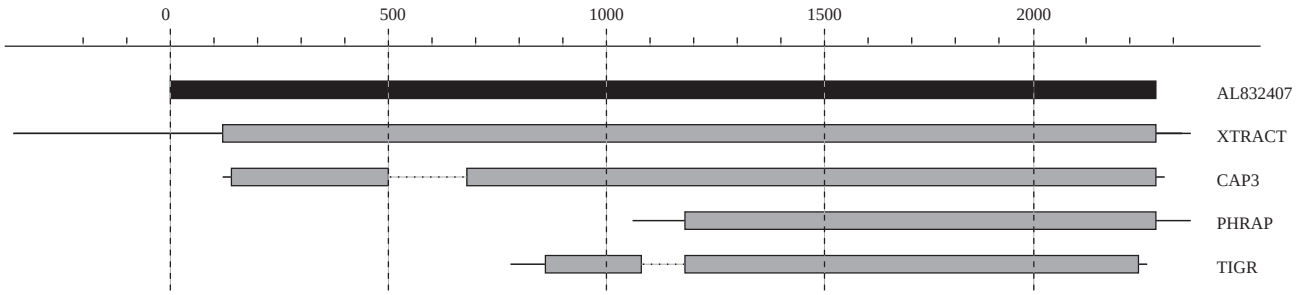

Fig. 5. Alignment of the best-matching consensus sequences against AL832407. The thick regions indicate matches, solid lines indicate non-matching regions, and stippled lines (in the CAP3 and TIGR outputs) indicate parts of the transcript that are missing from the consensus sequences.

to reconstruct most of the transcript, and CAP3 has a gap of 195 bases. The contig produced by xtract matches the reference exactly, CAP3 has 2 single-nucleotide errors, TIGR has 13, and Phrap has 17 single-nucleotide errors in the matched regions (not shown).

DISCUSSION AND CONCLUSIONS

We see from Figure 2 that Phrap and CAP3 produce a relatively low number of consensus sequences, while our algorithm and even more so, the TIGR assembler, produces a much larger amount. In many cases, TIGR is close to one contig for every two sequences in the data set. This likely reflects different goals; in particular, xtract will often generate different contigs for SNP variants in the data.

From Figure 3 we see that TIGR and Phrap are very fast, CAP3 is very slow, and our algorithm fluctuates in the middle, generally about twice as fast as CAP3.

The histogram in Figure 4 shows that in a majority of the cases (232 out of 272, or 85% of the total), our algorithm either produced the best contig or tied for the first place.

CAP3 is a clear second with highest-scoring contigs in 144 cases (53%). Among these were a surprisingly high number of ties (128 cases) where it scored the same as our algorithm, so it only succeeded in producing a *better* scoring contig in 16 cases. Both TIGR and Phrap score relatively poorly in our benchmark, with 25 (9%) and 29 (11%) first places, respectively (and roughly half of these were ties with another assembler, sometimes with all four, indicating clusters that are relatively easy to assemble correctly). This agrees well with the results published in Liang *et al*. (2000).

Since our algorithm will produce a set of transcripts so that every word in the input data with sufficient weight is contained in at least one consensus sequence, the number of sequences produced can be large. In order to get a set of consensus sequences more in line with the output of CAP3 and Phrap, we need to reduce the redundancy in the sequences produced by our algorithm. One way to do this is to use cd-hit (Li *et al*., 2001), which reduces a set of sequences to a set of representative sequences, eliminating sequences that have a high degree of similarity to the representatives.

As an experiment, we clustered the consensus sequences with *cd-hit*, and measured the quality and the number of contigs, as above. The results are presented in Figures 6 and 2. Running *cd-hit* was quite fast, and it processed the whole data set in about a minute.

We see that while the number of cases where our algorithm produces the best-scoring contig is reduced, our algorithm still outperforms CAP3, while the number of remaining contigs is more within the range of CAP3 and Phrap.

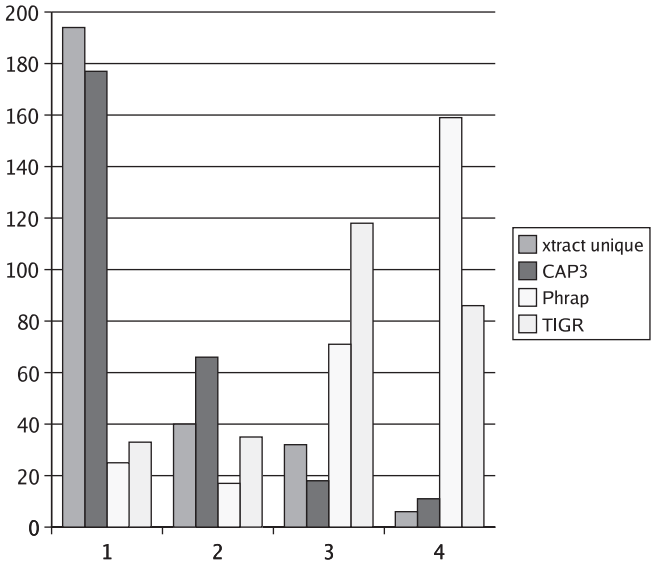

Fig. 6. The ranks of best contig, after processing redundant contigs with *cd-hit*.

While *cd-hit* is effective in removing a large amount of redundancy from the data set, it would probably be worthwhile to integrate this functionality in the algorithm itself, in order to be able to better identify potential SNPs, alternative splice variants and other transcript features.

The running time of the algorithm scales proportionally to the output sequence size (each graph traversal is linear in its length, with logarithmic factors in the graph size). Thus, generating sequences that later will be eliminated is costly, and a modified algorithm that ignores small sequence differences to produce a less redundant set of consensus sequences, should be much faster. As the current algorithm keeps track of where the generated consensus sequence matches the input data, it should be relatively easy to detect potential SNP sites as words occurring in multiple sequences matching a consensus well, but not themselves being part of the consensus.

We believe this experiment shows that this algorithm achieves better quality assemblies than more traditional approaches, with competitive running times.

ACKNOWLEDGMENTS

Phrap is available under an academic license. CAP3 is freely available for academic use, and was provided pre-compiled, along with advice on using it for EST clustering, by its author, Xiaoqiu Hang (pers. comm.). The TIGR Assembler can be downloaded from the TIGR web pages (http://www.tigr.org) as source code.

This work was funded by the Norwegian Salmon Genome Project, a Norwegian Research Council project.

REFERENCES

Batzoglou,S., Jaffe,D.B., 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.

Boguski,M. and Schuler,G. (1995) ESTablishing a human transcript map. *Nature Genet.*, **10**, 369–371.

Chevreux,B., Pfisterer,T., Drescher,B., Driesel,A.J., Miller,W.E., Wetter,T. and Suhai,S. (2004) Using the miraEST assembler for reliable and automated mRNA transcript assembly and SNP detection in sequenced ESTs. *Genome Res.*, **14**, 1147–1159.

Green,P. (1996) Phrap documentation, http://www.phrap.org/phredphrap/phrap.html

Heber,S., Alexeyev,M., Sze,S.-H., Tang,H. and Pevzner,P.A. (2002) Splicing graphs and EST assembly problem. *Bioinformatics*, **18**, S181–S188.

Huang,X. and Madan,A. (1999) CAP3: A DNA sequence assembly program. *Genome Res.*, **9**, 868–877.

Idury,R. and Waterman,M.S. (1995) A new algorithm for DNA sequence assembly. *J. Comput. Biol.*, **2**, 291–306.

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.

Li,W., Jaroszewski,L. and Godzik,A. (2001) Clustering of highly homologous sequences to reduce the size of large protein database. *Bioinformatics*, **17**, 282–283.

Liang,F., Holt,I., Pertea,G., Karamycheva,S., Salzberg,S.L. and Quackenbush,J. (2000) An optimized protocol for analysis of EST sequences. *Nucleic Acids Res.*, **28**, 3657–3665.

Malde,K., Coward,E. and Jonassen,I. (2003) Fast sequence clustering using a suffix array algorithm. *Bioinformatics*, **19**, 1221–1226.

Mironov,A., Fickett,J. and Gelfand,M. (1999) Frequent alternative splicing of human genes. *Genome Res.*, **9**, 1288–1293.

Modrek,B. and Lee,C. (2001) A genomic view of alternative splicing. *Nature Genet.*, **30**, 13–19.

Myers,E.W., Sutton,G.G., Delcher,A.L., Dew,I.M., Fasulo,D.P., Flanigan,M.J., Kravitz,S.A., Mobarry,C.M., Reinert,K.H. and Remington,K.A. (2000) A whole-genome assembly of drosophila. *Science*, **287**, 2196–2204.

Pertea,G., Huang,X., Liang,F., Antonescu,V., Sultana,R., Karamycheva,S., Lee,Y., White,J., Cheung,F., Parvizi,B., Tsai,J. and Quackenbush,J. (2003) Tigr gene indices clustering tools (tgicl): a software system for fast clustering of large est datasets. *Bioinformatics*, **19**, 651–652.

Pevzner,P.A., Tang,H. and Waterman,M.S. (2001) An Eulerian path approach to DNA fragment assembly. *PNAS*, **98**, 9748–9753.

Skiena,S.S. and Sundaram,G. (1993) Reconstructing strings from substrings. *Lecture Note. Comput. Sci.*, **709**, 565–576.

Staden,R. (1996) The Staden sequence analysis package. *Mol. Biotechnol.*, **5**, 233.

Sutton,G., White,O., Adams,M. and Kerlavage,A. (1995) TIGR assembler: A new tool for assembling large shotgun sequencing projects. *Genome Sci. Technol.*, **1**, 9–19.