Genome Sequence Assembly Using Trace Signals and Additional Sequence Information

Chevreux B.[1]*, Wetter, T.[2] and Suhai, S.[1]

[1] Department of Molecular Biophysics,
German Cancer Research Centre Heidelberg, 69120 Heidelberg, Germany

[2] Institute for Medical Biometry and Informatics
University of Heidelberg, 69120 Heidelberg, Germany

Abstract

Motivation: This article presents a method for assembling shotgun sequences which primarily uses high confidence regions whilst taking advantage of additional available information such as low confidence regions, quality values or repetitive region tags. Conflict situations are resolved with routines for analysing trace signals.
Results: Initial tests with different human and mouse genome projects showed promising results but also demonstrated the need to recognise and handle correctly very long, untagged and non-standard repeats.
Availability: Current versions of the MIRA assembler are available on request at the canonical project homepage as binary for SGI, Intel Linux and SUN Solaris: http://www.dkfz-heidelberg.de/mbp-ased/
Contact: b.chevreux@dkfz-heidelberg.de

Introduction

Todays large scale genome sequencing efforts produce enormous quantities of data each day. They are now nearly all based on the chain-termination dideoxy method published by Sanger et al. (1977) in one way or another. But the gel or capillary electrophoresis used can determine only about a maximum of 1000 to 1500 bases, the high quality stretch with low error probabilities for the called bases often being around the first 400 to 500 bases. Current sequencing strategies for a contiguous DNA sequence (contig) – ranging anywhere between 20 kilobases (kb) and 200 kb – therefore basically boil down to fragment the given contig in hundreds or thousands of overlapping subclones (Durbin and Dear (1998)), analyse these by electrophoresis and subsequently assemble the subclones back together in one contig.

The extensively studied reconstruction of the unknown, correct contiguous DNA sequence by inferring it through the help of a number of representations[1] is called the assembly problem. The devil is in the details, however. If the collected readings

*To whom correspondance should be adressed

[1] also called fragments, see Myers (1995)

(reads) were 100% error free, then a multiplicity of problems would not occur. In reality, the extraction of data by gel electrophoresis is a physical process in which errors due to chemical artifacts like compressions show up quite often. Ewing et al. (1998); Ewing and Green (1998) show that – together with errors occuring in the subsequent signal analysis – current laboratory technologies total an error rate that might be anywhere between 0.1% – for good parts in the middle of a read – and more than 10% in bad parts of a read at the very beginning and at the end. This error rate, combined with the sometimes exacerbating fact that DNA tends to contain highly repetitive stretches with only very few bases differing across different repeat locations, impedes the assembly process in an awesome way.

The above mentioned error rates and repetitive properties of DNA lead to the necessity of using fault tolerant and alternatives-seeking algorithms. Wang and Jiang (1994) showed that the assembly problem – even using error free representations (fragments) of the true sequence – is NP complete. This means that the volume of data can only be assembled by approximating strategies, relying on algorithms that are well-behaved in time and space complexity.

Assembly strategies

Referring to Dear et al. (1998), a ”sequence assembly is essentially a set of contigs, each contig being a multiple alignment of reads”. A number of different strategies have been proposed to tackle the problem, ranging from simple greedy pairwise alignments – sometimes using additional information (Peltola et al. (1984)), sometimes using a whole set of refinements (Huang (1996)) – to weak AI methods like genetic algorithms (Parsons et al. (1993); Notredame and Higgins (1996); Zhang and Wong (1997)).

There are nowadays mainly two different existing approaches for assembling sequences: (i) the iterative and (ii) the all-in-one-step approach. The first type of assembly is essentially derived from the fact that the data analysis and reconstruction approximation algorithms can be parametrised differently, ranging from very strict assembly of only the highest quality parts to very ’bold’ assembly of even lowest quality stretches. An assembly starts with strictest parameters, having the output edited manually (by highly trained personnel) or by software and then the process is reiterated with less strict parameters until the assembly is finished or the parameters become to lax (see figure 1). The second approach has been made popular by the PHRAP assembler presented by Phil Green[2]. This assembler uses low and high quality sequence data from the start and generates a consensus by puzzling its way through an assembly using the highest quality parts as reference, giving the result to a human editor for finishing.

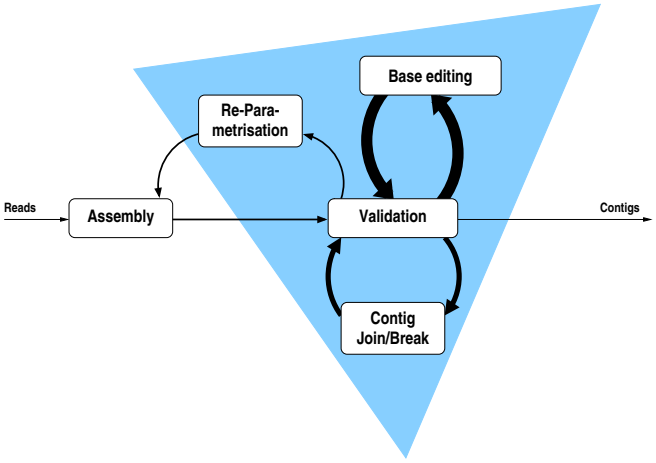

Figure 1: Conventional assembly has a high level of human interaction (area in the triangle). The thickness of the arrows represents the relative number of times a certain action has to be performed.

A common characteristic to all existing assem-

[2]http://www.phrap.org/

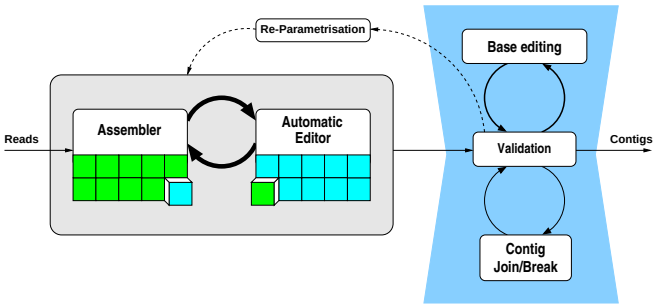

Figure 2: Using an integrated assembly and editing concept transfers a significant portion of the work to automated algorithms, leaving only non-standard problems to the human finisher (area in the hexagon).

blers is that they rely on the quality values with which the bases have been attributed by a base caller. Within this process, an error probability is computed by the base caller to express the confidence with which the called base is thought to be the true base. The positive aspect is the possibility for assemblers to decide in favour of the best, most probable bases when a discrepancy occurs. The negative aspect of current base callers is their inability to write confidence values for optional, uncalled bases at the same place. This could improve the search for alternative assemblies substantially.

We therefore implemented a third type of assembler, trying to combine and substantially extend the strengths of both approaches mentioned above and copying assembly analysis strategies done by human experts (see figure 2). An important criterion in the design of our assembler is the quality aspect of the final result: the assembler works only with the stretches of DNA sequences marked as 'high' or 'acceptable' quality, from which it selects the best to start an assembly. These high confidence regions (HCR) ensure a firm base and good building blocks during the assembly process. Lower quality parts (low confidence regions, LCR) can be used later on if needed.

But the main difference of our approach is that we combined the assembler with some capabilities of an automatic editor. Both the assembler and the automatic editor are separate programs and run separately, but we view the task of assembly and finishing to be closely related enough for both parts to include routines from each other (see also Pfisterer and Wetter (1999)). In this process, the assembler gains the ability to perform signal analysis on partly assembled data which helps to reduce misassemblies especially in problematic regions like repeats (ALUS, REPT etc.), where simple base qualities alone could not help. Analysing trace data at precise points with a given hypothesis 'in mind'[3] is a substantial advantage of a signal analysis aided assembler compared with a 'sequential base caller and assembler' strategy, especially while discriminating alternative solutions during the assembly process. In return, the automatic finisher gains the ability to use alignment routines provided by the assembler.

Methods and Algorithms

We worked out a multi-phased concept to have our assembler perform the difficult task of shotgun sequence alignments. Different authors have proposed different sets of acceptance criteria for the optimality of an alignment (Chan et al. (1992)). Traditionally (Myers (1995)), "the objective of this (assembly) problem has been to produce the shortest string that contains all the fragments as substrings, but in case of repetitive target sequences this objective produces answers that are overcompressed." As a result to this, we conceived the strategy of the 'least number of unexplainable errors' present in an assembly to be optimal.

To demonstrate the working principles throughout this document, we will use a toy project of six

[3] e.g. 'could the base A at position 235 in read 1 be replaced by a G?' (because the overall consensus at this position of the other reads suggests this possibility)

reads – of which we assume to have an overall fair base quality – as example.

Data preprocessing

A high confidence region (HCR) of bases within every read is to be selected as an anchor point for the next phases. Existing base callers (ABI, PHRED and others) detect bases and rate their quality quite accurately and keep increasing in their performance, but bases in a called sequence always remain afflicted by increasing uncertainty towards the end of a read. This additional information, potentially worthful, can nevertheless constitute an impeding moment in the early phases of an assembly process, bringing in too much noise or preventing the correct determination of true, long range repeats.

Another important factor is sequencing vector, which will invariably be found at the start of each read. This earliest part of any cloned sequence must be marked or removed from the assembly. In analogy to the terms used in the GAP4 package, we will refer to LCR also as 'hidden' data (Staden et al. (1997)) whereas sequence marked as being part of the sequencing vector is 'marked' data.

This is the information the assembler will work with, any of which can be left out (except sequence and vector clippings) but will reduce the efficiency of the assembler:

1. the initial trace data, representing the gel electrophoresis signal
2. the called DNA sequence
3. position specific confidence values for the called bases of the DNA sequence
4. a stretch of DNA in each sequence marked as HCR
5. general properties like name of the sequencing template etc.
6. special DNA properties in different regions of a read (like sequencing vector, standard repeat sequence etc.) that have been tagged or marked

The data preprocessing step has been taken out of the actual assembler as almost every laboratory has its own means to define 'good' quality within reads and already use existing programs to perform this task. For example, quality clipping, sequencing vector and cosmid vector removal are controlled by the PREGAP script provided with the GAP4 package (Bonfield et al. (1995); Staden (1996); Bonfield and Staden (1996)) or can be done with cross_match[4] provided by PHRAP.

Read scanning

A common start for an assembly is to compare every read with every other read (and its reversed complement) using a fast and fault-tolerant algorithm to detect potential overlaps. We developed an algorithm based on the Shift-AND text search algorithm introduced by Wu and Manber (1992b,a) which extended ideas of Baeza-Yates and Gonnet (1992). The Shift-AND algorithm allows – with a configurable error threshold – two sequences to be recognised as partly identical only by shifting bit vectors. The type of errors within the partial identity is irrelevant as insertions, deletions and mismatches in sequences are equally recognised. Originally, the complexity of the Shift-AND algorithm allowing errors is $O(cnm)$ (Gusfield (1997)), where c is a constant depending of the number of errors allowed in the match, m being the length of the pattern and n is the length of the sequence to be compared with. Our modified implementation of the algorithm has the complexity $O(cn)$ as we are able to have $O(m) = 1$ by using a limited number[5] of small patterns that fit into single machine

[4] http://www.phrap.org/phrap.docs/general.html

[5] i.e. a minimum of 3 and a mximum of 5

registers.

The efficiency of the algorithm has been demonstrated by testing it on simulated shotgun data sets of real world DNA sequences and comparing our results with previously published ones by Huang (1996). We found that our fast scanning method is especially well suited for finding weak overlaps or overlaps in error-rich regions. For example, the DNASAND algorithm was always able to find potential overlaps with a false positive rate below 0.5 and with less than 2% of missed overlaps, even in data sets with an artificially introduced high error rate of 10%. These rates are lower than previously published results. Full results and implementational detail for our DNASAND algorithm are in preparation to be published (Chevreux et al. (1999)).

Although the DNASAND algorithm does not specify the overall type of global relationship of two sequences (total correspondence, containment and overlapping, see Huang (1994)), any type of this relationship is recognised. As result of this first scan, a matrix containing information on potential overlaps of all the fragments is generated and the direction of the potential overlap (forward–forward or forward–complement) is generated. As shown in figure 3, there really are two separate matrices as – theoretically – a read can overlap with another read in both directions.

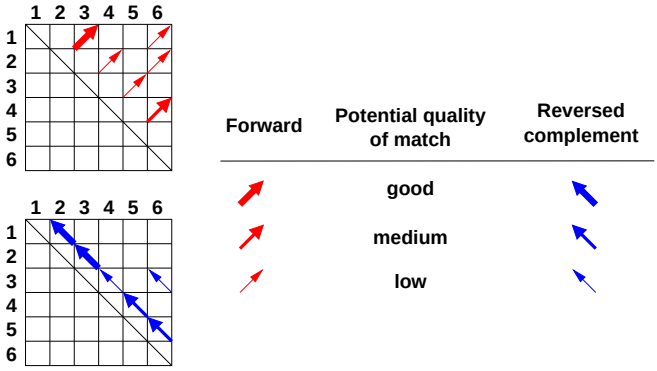

Figure 3: The matrices generated after the first fast scan of every read against every other in search for potential overlaps. Unlike this small example with 6 reads might suggest, the matrices in real world projects – with a large number of reads – are normally sparsely occupied.

Systematic match inspection

In the next fundamental step, potential overlaps found during the scanning phase are examined with a Smith-Waterman based algorithm for local alignment of overlaps. Our SW alignment algorithm takes into account that, regarding the fact that todays base caller have a low error rate, the block-indel model (Giegerich and Wheeler (1996)) does not apply to the alignment of shotgun sequencing data. This means that long stretches of mismatches or gaps in the alignment are less probable than small, punctual errors. We also assume a 'mismatch' of a base against a 'N' (symbol for aNy base) to have no penalty as in many case the base caller rightfully set 'N' for a real existing base (and not an erroneous extra one) that could not be resolved further.

Quality criteria like the score of the expected length of the overlap, the overall computed score of the overlap etc. are calculated for each overlap. Every candidate pair whose computed score is within a configurable threshold of the expected score and where the length of the overlap is not too small is accepted as 'true' overlap, candidate pairs not matching these criteria – often due to spurious hits in the scanning phase – are identified and rejected from further assembly (see figures 4 and 5).

The aligned dual sequences (ADS) – along with complementary data (like orientation of the aligned reads, overlap region etc.) – that passed the Smith-Waterman test are stored to facilitate and speed up the next phases. Good alternatives are also stored to enable alternative alignments to be found later on in the assembly.

All the ADS form one or several weighted graphs

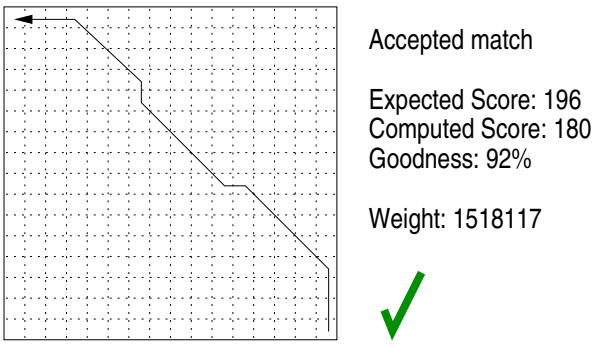

Figure 4: A modified Smith-Waterman algorithm for local alignment is used to confirm or reject potential overlap found in the fast scanning phase. Accepted overlap get a weight assigned depending on the length of the overlap and alignment quality.

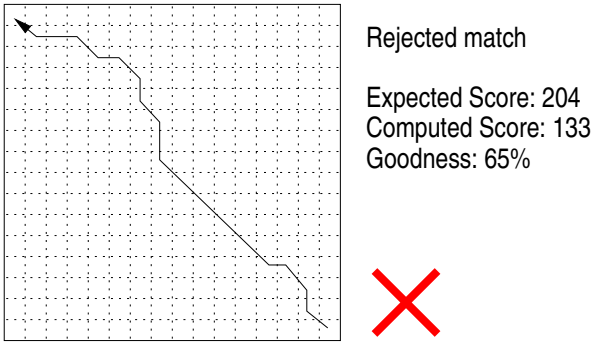

Figure 5: Although having a good partial score, the overall goodness of this alignment is too low to be accepted. This read pair is eliminated from the list of possible overlaps.

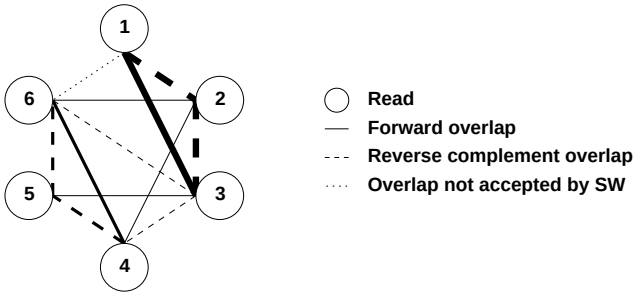

Figure 6: An overlap graph generated from the aligned overlaps that passed the Smith-Waterman test. The thickness of the edges represents the weight of an overlap. Although the 1–6 overlap is marked for demonstration purposes in this figure, rejected overlaps (due to spurious hits in the scanning phase, see figure 3) are not present in the graph.

which represent the totality of all the assembly layout possibilities of a given set of shotgun sequencing data (see figure 6). The nodes of the graph are represented by the reads. An edge between two nodes indicates that these two reads are partially overlapping. The weight of the edges themselves are computed from the squared quality of the alignment multiplied with the length of the overlap. This emphasises the quality aspect of the alignment and also takes the length of the overlap into account.

Building contigs

The overlaps found and verified in the previous phases must then be assembled into contigs. This is the most fundamental and intricate part of the process, especially in projects containing many repetitive elements. Several basic approaches to the multiple alignment problem have been devised to tackle this problem. Although algorithms for aligning multiple sequences at once have been used with increasing success lately for up to about 10 to 15 sequences (Stoye (1998)), the amount of time needed

to perform this alignment is still too unpredictable[6] to be used in sequence assembly.

We decided to use iterative pairwise sequence alignment and devise new methods for searching overlap candidates and for empowering contigs to accept or reject reads presented to them during the contig building. The algorithm consists mainly of two objects which interact with each other: a pathfinder module and a contig building module.

Pathfinder and contig interaction

Because we use an iterative approach to the multiple alignment problem – this means we always successively align an existing consensus against the next read – the results of the alignment sensitively depends on the order of pairwise alignments (Morgenstern et al. (1996)). We have to make sure that we start at the position in the contig where we mostly have many reads with almost no errors, the pathfinder will thus – in the beginning – search for a node in the weighted graph having a maximum number of highly weighted edges to neighbours. The idea behind this behaviour is to take the read with the longest and qualitatively best overlaps with as many other reads as possible. This ensures a good starting point: the 'anchor' for this contig.

We first tried a simple greedy algorithm to determine the next overlap candidate to a contig, but on occasions – especially in highly repetitive parts of a genome – this algorithm fails. With our current algorithm, the number of misalignments could be substantially reduced: the pathfinder designates the next read to add to an existing contig by making an in-depth analysis[7] of the weights to neighbouring reads, taking the edge leading to the first node that is contained in the best partial path found so far. It then presents this read (and its approximate position) to the contig object as potential candidate for inclusion into the existing consensus.

[6] ranging from a few seconds to more than one hour

[7] with a cutoff value of normally 4 or 5 recursions

A contig is represented by a collection of reads that have been arranged in a certain order with given offsets to form an alignment that is as optimal as possible, i.e. an alignment where the reads forming it have as few unexplained errors as possible but still form the shortest possible alignment. To serve this purpose, a contig object has been provided with functions that analyse the impact of every newly added read (at a given position) on the existing consensus. Our assumption is now that – as the assembly started with the best overlapping reads available – the bases in the consensus will be right at almost every position. Should the newly added read integrate nicely into the consensus, perhaps extending it, then the contig object will accept this read as part of the consensus. In cases representing a chance alignment, the read differs in too many places from the actual consensus (thus differing in many aspects from reads that have been introduced to the consensus before). The contig will then reject the read from its consensus and tell the pathfinder object to search for alternative paths through the weighted overlap graph. The pathfinder object will eventually try to add the same read to the same contig but at a different position or – skipping it – try other reads.

Once the pathfinder has no possibilities left to add unused reads to the actual contig, it will again search for a new anchor point and use this as starting point for a new contig. This loop continues until all the reads have been put into contigs or – if some reads could not be assembled anywhere – form single-read contigs[8].

Consensus approval methods

As mentioned briefly above, each contig object has the possibility to reject a read being introduced into

[8] Of course, a single read itself cannot be called a contig. But putting it into the same datastructure (a contig object) like the other, assembled reads is a convenient way to keep unassembled reads in a database.

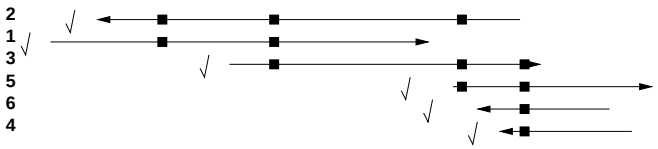

Figure 7: Example of simplest possible assembly which does not take advantage of additional knowledge. Reads are assembled as the algorithm walks through the path, the checkmarks showing the reads have been accepted as matching to the contig. Black squares mark discrepancy columns not having the same bases.

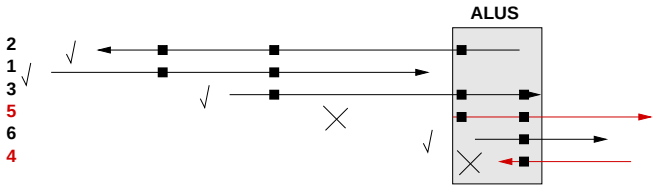

Figure 8: Using additional information on standard repetitive stretches, the contig object checks the repeat regions more thoroughly and rejects reads that have too many errors that cannot be explained by signal analysis.

its consensus. This is one of the most crucial points in the development of our assembler: allowing presumably good building blocks, i.e. reads with high quality, to start an assembly is a decisive step in the ongoing assembly process. It allows to use implicit and explicit knowledge available in reads that are known to be good.

The simplest behaviour of a contig could be to simple accept every new read that is being presented as part of the consensus without further checks. In this case the assembler would thus rely only on the weighted graph, the method with which the weighted edges were calculated and the algorithm which traverses this graph. Figure 7 shows this exemplarily for the reads from the example we set up above. Although there are four columns having different bases, the assembly looks quite reasonable.

However, using additional information that is available at the time of assembly might prove useful. For example, known standard repetitive elements have been tagged in the data preprocessing step of the assembly. It is therefore possible to apply much stricter control mechanisms in those stretches of a sequence known to be repetitive, because they might really differ, but only in a few bases and are thus dangerous in the assembly.

Figure 8 shows this information – along with a new assembly that took place – using the knowledge that repetitive elements must be checked much more strictly. In this new assembly, read 5 and read 4 were rejected from the contig as the pathfinder tried to enter them, because they induced errors into the already existing contig formed by the reads 2-1-3 (when adding read 5) and 2-1-3-6 (when adding read 4).

The contig object failed to find a valid explanation to this problem when calling the signal analysis function – provided by the automatic editor – that tried to resolve discrepancies by investigating probable alternatives at the fault site. This labour intensive task is normally executed by highly trained personnel although in a significant number of cases it is fairly easy to adjudicate between conflicting readings by analysing the trace data. A previous suggestion on incorporating electrophoresis data into the assembly process promoted the idea of capturing intensity and characteristic trace shape information and provide these as additional data to the assembly algorithm (Allex et al. (1996)). We decided against such an approach as essentially, all the assembler – and with it the consensus finding algorithm of a contig – needs to know is if yes or no the signal analysis reveals enough evidence for resolving a conflict within reads by changing the bases incriminated. Signal analysis is therefore treated as a black box decision formed by an expert system only called during the assembly algorithm when conflicts arise. It provides nevertheless

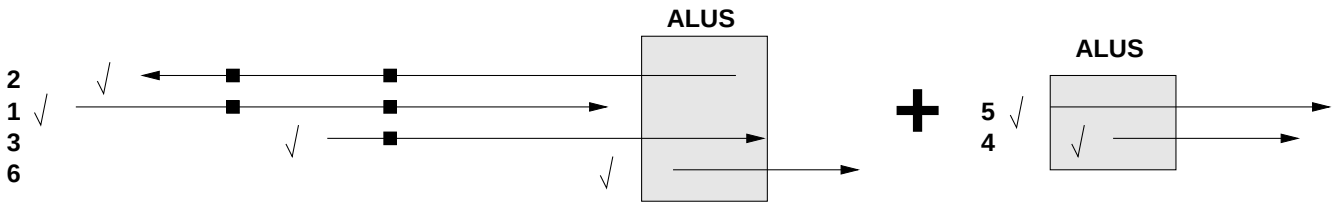

Figure 9: Using strict signal checking in the ALUS repeat region, unexplained errors as shown in figure 8 lead to the formation of a second contig. No unexplained errors remain in the region known as dangerous.

more information than the one contained in quality values extracted from the signal of one read only (compare to Durbin and Dear (1998)): there is a reasonable suspicion deduced from other aligned reads on the place and the type of error where the base caller could have made a mistake.

Figure 9 shows the result of the assembly that began in figure 8. There is not enough evidence found in the conflicting reads to allow a base change to resolve the discrepancies arising. There is also the additional knowledge that these non-resolvable errors occur in an area tagged as 'ALU-repeat', which leads to a highly sensitive assessment of the errors. Consequently, the reads 4 and 5 – containing repetitive sequence which the first contig object could not make match to its consensus – form a second contig as shown in the example.

Read extension

As the initial assembly used only high quality parts of the reads, further information can be extracted from the assembly by examining the end of the reads that were previously unused because the quality seemed too low. Although the signal-to-noise in read traces quickly degrades toward the end, the data is not generally useless. These 'hidden' parts of the reads can now be uncovered in two ways: (i) by uncovering parts of the reads that align to the already existing consensus and (ii) by uncovering hidden stretches of reads at the end of the contigs that are not confirmed by a consensus.

Intracontig extension is used to uncover reads and 'beef up' areas of low coverage within a contig and is a straightforward process. Mainly used as a method to get more data confirmation than is available using only high quality parts, the hidden sequence is aligned step by step to the existing consensus. In most cases, the discrepancies found between the HCRs forming the existing consensus and the unaligned LCR will be decided in favour of the HCR. But in some cases, especially in regions with very low coverage, one or more reads with LCR data can correct an error in the HCR stretch, e.g. when there is a local drop in the confidence values and signal quality of bases in the HCR stretch whereas signal quality and confidence values of the same bases in the LCR stretch seem better.

Extracontig read extension, the second possibility, uncovers LCR at the ends of contigs and is used to extend the consensus to the left or to the right of a contig. LCR data present at the end of probably each read is not directly bad quality, but it is treated as hidden data: a region where the base caller calculated lower quality for the bases because it depended on the trace data of a single read. However, once reads have been aligned in their HCR, two or more stretches of lower quality can be used to uncover each other. The main purpose for this is to enable potential joins between contigs to be made.

The iterative enlargement procedure enables the assembler to redefine step by step the HCR of each

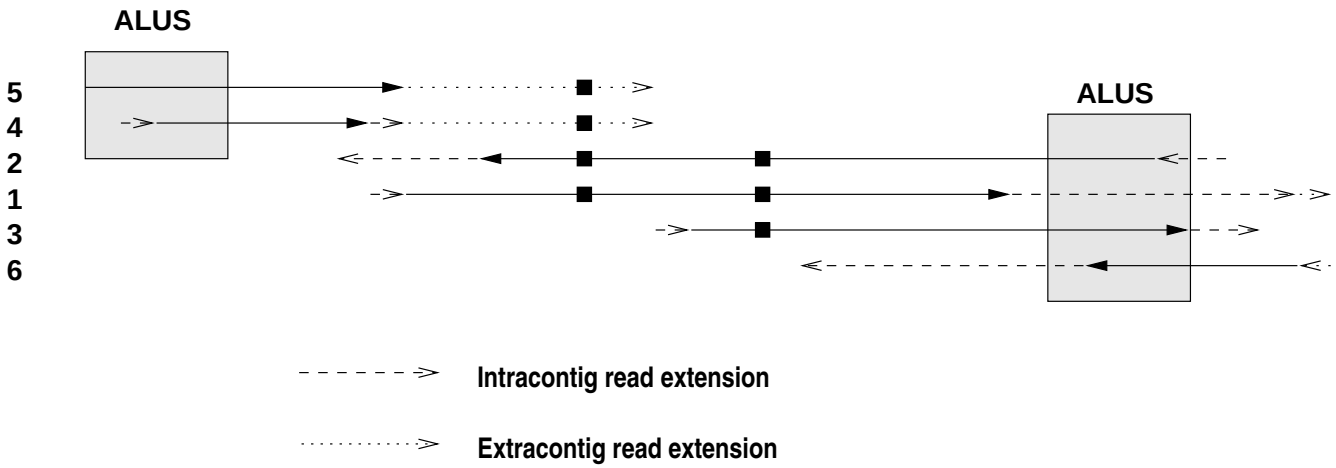

Figure 10: Intracontig read extension increases the coverage of contigs while extracontig read extension allows existing contigs to be joined after the assembly process. After extracontig extension, the contigs could be joined.

read by comparing it with supporting sequences from aligned reads. This use of information in collateral reads is the assembler's major advantage over a simple base caller, which has only the trace information of one read to call bases.

Contig linking and editing

As last step of the assembly, the extended contigs are linked together where possible and the result is passed to the automatic finisher to correct errors in the assembly. Depending on the number and type of errors found in the assembled data by the finisher, the contigs can be dismantled and re-assembled, taking into account the corrections made to the individual reads by the autofinisher and therefore refining the overall assembly.

By cycling through the previous steps, the assembler iteratively corrects errors – like misassembled repetitive repeats – that were made during previous steps and thus ensures the resulting contigs contain as few unexplainable errors as possible.

Results and discussion

The principles presented within this paper have been implemented in our MIRA assembler. MIRA is now in use at the IMB Jena Genome Sequencing Centre after having passed an intensive testing phase in late March 1999, during which different bugs were removed from the program and the overall concept has been refined. We tested our assembler against finished projects by re-assembling with the original data and comparing the result with the finished contig. The most difficult case was a project of approximately 140kb in 3942 files, containing 47 spatially separated ALU sites (331 files had an ALU tag set).

Using only the HCR without intra- and extracontig read extension algorithms[9], the assembler produced 18 contigs of length $>$ 1000 bases which covered about 97% of the finished project. We found only one case of an actual misassembly by visual inspection. It showed to be a very weak

[9]read extension and contig join algorithms were not present in late March and are still under development as of the time of this writing

overlap of approximately 40 bases in an ALU region, where some of the reads had not been correctly tagged as ALU, thus inhibiting the MIRA assembler to correctly check and qualify this region as dangerous zone with strict checking rules. On the other hand, using MIRA with stricter Smith-Waterman align parameters than the standard parameters would have prevented this.

No other crucial problems arised, MIRA delivered an otherwise correct assembly with significantly less contigs than the standard gap4 cycle assembly. This showed us that our approach of assembling HCR and strictly checking stretches of DNA known as problematic with routines for signal analysis avoids crucial mistakes in the assembly while preparing good contigs for possible join operations that are to be performed later.

Conclusion and further work

We presented a new method for assembling shotgun sequence data, which combines strengths of existing assemblers and extends them with the ability to use additional knowledge present in the data and to resolve conflicts during the assembly by falling back to trace signal analysis routines. Initial test showed promising results, but usage of the MIRA assembler on a daily basis shows that methods to recognise and correctly assemble very long, non-standard and thus untagged repeats are needed. We plan to implement such coverage and clustering routines and integrate them into our pathfinder and contig objects. The current beta version of the assembler is available as binary for SGI, Intel Linux and SUN Solaris platforms on request at the authors address.

Acknowledgements

The authors would like to thank Thomas Pfisterer for rigorously reviewing a draft of this paper. We also want to acknowledge and thank the people at the genome sequencing group of the IMB Jena and the LION AG Heidelberg for their patience and numerous enhancement tips whilst explaining the mysteries of DNA and the shotgun sequencing process.

References

Allex, C. F., Baldwin, S. F., Shavlik, J. W. and Blattner, F. R. (1996), Improving the Quality of Automatic DNA Sequence Assembly using Fluorescent Trace-Data Classifications. *ISMB*, **4**, 3–14.

Baeza-Yates, R. A. and Gonnet, G. H. (1992), A New Approach to Text Searching. *Commun. ACM*.

Bonfield, J. K., Smith, K. F. and Staden, R. (1995), A new DNA sequence assembly program. *Nucleic Acids Research*, **23**(24), 4992–4999.

Bonfield, J. K. and Staden, R. (1996), Experiment files and their application during large-scale sequencing projects. *DNA Sequence*, **6**, 109–117.

Chan, S. C., Wong, A. K. C. and Chiu, D. K. Y. (1992), A survey of multiple sequence comparison methods. *Bulletin of Mathematical Biology*, **54**(4), 563–598.

Chevreux, B., Wetter, T. and Suhai, S. (1999), Effective Filtering of Genomic Sequences. in preparation.

Dear, S., Durbin, R., Hilloier, L., Marth, G., Thierry-Mieg, J. and Mott, R. (1998), Sequence Assembly with CAFTOOLS. *Genome Research*, **8**, 260–267.

Durbin, R. and Dear, S. (1998), Base Qualities Help Sequencing Software. *Genome Research*, **8**, 161–162.

Ewing, B. and Green, P. (1998), Base-Calling of Automated Sequencer Traces Using Phred. II. Error Probabilities. *Genome Research*, **8**, 186–194.

Ewing, B., Hillier, L., Wendl, M. C. and Green, P. (1998), Base-Calling of Automated Sequencer Traces Using Phred. I. Accuracy Assessment. *Genome Research*, **8**, 175–185.

Giegerich, R. and Wheeler, D. (1996), Pairwise Sequence Alignment. http://www.techfak.uni-bielefeld.de/bcd/Curric/PrwAli/prwali.html.

Gusfield, D. (1997), *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press.

Huang, X. (1994), On global sequence alignment. *Computer Applications in the Bioscience*, **10**(3), 227–235.

Huang, X. (1996), An Improved Sequence Assembly Program. *Genomics*, **33**, 21–31.

Morgenstern, B., Dress, A. and Werner, T. (1996), Multiple DNA and protein sequence alignment based on segment-to-segment comparison. *Proceedings of the National Academy of Science USA*, **93**, 12098–12103.

Myers, E. W. (1995), Toward Simplifying and Accurately Formulating Fragment Assembly. *Journal of Computational Biology*, **2**(2), 275–290.

Notredame, C. and Higgins, D. G. (1996), SAGA: sequence alignment by genetic algorithm. *Nucleic Acids Research*, **24**(8), 1515–1524.

Parsons, R., Forrest, S. and Burks, C. (1993), Genetic Algorithms for DNA Sequence Assembly. *ISMB*, 310–318.

Peltola, H., Söderlund, H. and Ukkonen, E. (1984), SEQAID: a DNA sequence assembling program based on a mathematical model. *Nucleic Acids Research*, **12**(1), 307–321.

Pfisterer, T. and Wetter, T. (1999), *Computer Assisted Editing of Genomic Sequences - Why and How We Evaluated a Prototype*, Springer-Verlag, Berlin Heidelberg New York. Lecture Notes in Artificial Intelligence; Subseries of Lecture Notes in Computer Science, pp. 201–209.

Sanger, F., Nicklen, S. and Coulson, A. (1977), DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Science USA*, **74**, 5463–5467.

Staden, R. (1996), The Staden Sequence Analysis Package. *Molecular Biotechnology*, **5**, 233–241.

Staden, R., Bonfield, J. and Beal, K. (1997), *The New Staden Package Manual - Part 1*. Medical Research Council, Laboratory of Molecular Biology.

Stoye, J. (1998), Multiple sequence alignment with the divide-and-conquer method. *Gene / GC*, **211**, 45–56.

Wang, L. and Jiang, T. (1994), On the Complexity of Multiple Sequence Alignment. *Journal of Computational Biology*, **1**(4), 337–348.

Wu, S. and Manber, U. (1992a), Approximate Pattern Matching. *Byte Magazine*, **11**, 281–292.

Wu, S. and Manber, U. (1992b), Fast Text Searching Allowing Errors. *Commun. ACM*, **35**(10), 83–91.

Zhang, C. and Wong, A. K. (1997), A genetic algorithm for multiple molecular sequence alignment. *Computer Applications in the Bioscience*, **13**(6), 565–581.