Chaining Algorithms for Alignment of Draft Sequence

Mukund Sundararajan[1], Michael Brudno[1], Kerrin Small[2],
Arend Sidow[2,3], and Serafim Batzoglou[1]

[1] Stanford University, Department of Computer Science, Stanford, California, 94305 USA
{mukunds,brudno,serafim}@CS.Stanford.edu
[2] Stanford University, Department of Genetics, Stanford, California, 94305 USA
{kerrin,sidow}@Stanford.edu
[3] Stanford University, Department of Pathology, Stanford, California, 94305 USA

Abstract. In this paper we propose a chaining method that can align a draft genomic sequence against a finished genome. We introduce the use of an overlap tree to enhance the state information available to the chaining procedure in the context of sparse dynamic programming, and demonstrate that the resulting procedure more accurately penalizes the various biological rearrangements. The algorithm is tested on a whole genome alignment of seven yeast species. We also demonstrate a variation on the algorithm that can be used for co-assembly of two genomes and show how it can improve the current assembly of the *Ciona savignyi* (sea squirt) genome.

1 Introduction

In bioinformatics, the development of novel biological techniques has created new computational challenges. This is perhaps best epitomized in the problem of sequence alignment, where the development of new sequencing techniques has led to the demand for alignment algorithms capable of dealing with mega-base long regions while maintaining high sensitivity. While in the past it has been sufficient to use two general types of alignment methods, global (Needleman and Wunsch, 1970) and local (Smith and Waterman, 1982), both approaches have been shown recently to be insufficient for alignment of long genomic sequences that have undergone rearrangements (Brudno et al, 2003c). Alignment of draft sequence, where one or both of the sequences being aligned is split into a set of unordered contigs creates similar problems. Global algorithms cannot handle draft sequence at all, while local algorithms just report all the similarity that they find and do not reproduce the syntenic regions that exist between genomes. For many of the genomes currently being sequenced, there are no plans to finish the sequence, and as the number of draft genomes grows, there will be an increasing need for algorithms that can effectively align such sequences.

Unfortunately, the problem of ordering a set of DNA sequences based on another set of (possibly unordered) DNA sequences has been shown to be MAX-SNP hard (Veeramachaneni et al 2002), even in the simpler case of ordering a set of contigs against a finished chromosome. To our knowledge Avid (Bray et al. 2003) and MUMmer 2.0 (Delcher et al. 2002) were the first programs to order a draft sequence based on a second, "finished" sequence based on effective heuristics.

I. Jonassen and J. Kim (Eds.): WABI 2004, LNBI 3240, pp. 326–337, 2004.

Recently, sparse dynamic programming based chaining techniques have come to the forefront as a successful approach for fast sequence alignment. Typically, a local aligner produces sections of homology, called fragments. Chaining involves selecting a high scoring subset of these based on some objective criteria. Since the original demonstration that fragment chaining could be achieved in $O(n \lg n)$ time by Eppstein and colleagues (Eppstein et al. 1992) the method has been used to speed up global alignment (Delcher et al 1999). Recently Abouelhoda and Ohlebusch (2003) and Brudno et al. (2003c), have used different variations of the sparse dynamic programming algorithm to find rearrangements between genomes. Lippert and colleagues have used a variation on the Brudno et al. (2003c) algorithm to find the differences between two assemblies of a genome (Lippert et al. 2004).

In this work we use a sparse dynamic programming chaining algorithm to align and co-assemble draft genomes. In particular, we make two important modifications to the chaining with rearrangements algorithm (Brudno et al. 2003c, Lippert et al. 2004). An *unrelated* gap penalty is introduced, to chain fragments that are on different contigs of the draft sequence. This formulation allows setting a threshold for the minimal sequence similarity necessary to include a contig in the ordering. We also introduce a new model for sparse dynamic programming that allows us to efficiently make intelligent chaining decisions and enforce penalties based not only on the last local alignment in the chain, but also on previous fragments. Though the solution is no longer optimal with respect to the fore defined objective criteria, this technique improves the chains formed in practice.

We test our draft alignment algorithm on sequences of the seven yeast genomes (Kellis et al. 2003, Cliften et al. 2003). The finished *Saccharomyces cerevisiae* genome was compared to four other yeasts of the sensu-stricto group, one from sensu-lato, and one petite-negative yeast. While we are able to align accurately the four genomes of sensu-stricto, the alignment of the two more distant genomes is less reliable, both indicating the necessity for future work, and suggesting the extent to which draft genomic sequence can be used to shed light on a distant relative.

We have also applied our chaining technique to the problem of co-assembling a genome. In this problem one is given two different assemblies, in draft form, of the same genome, with each assembly organized into contigs (contiguous pieces of the DNA sequence), which are joined into scaffolds (orderings of contigs) by the assembly program based on the paired-read information. When one is given two assemblies of the same data, it becomes possible to correct the potential errors in each assembly, fill the gaps between contigs and label contigs that are potentially misassembled. Here we present a practical method for sequence co-assembly and test it on 19 regions constituting 10% of the *Ciona savignyi* genome. Our method was able to identify more than 60 potential misassemblies with only one false positive, and in all but two cases correctly ordered the contigs to build an assembly with larger contigs and scaffolds.

2 Draft Sequence Alignment

The goal of the draft sequence alignment problem is to map the sections of the finished sequence (Sequence 1) to sections of the draft contigs (Sequence 2). This synteny map is used to construct a reference alignment with respect to the finished

sequence. We present an algorithm to solve this problem based on the Sparse Dynamic Programming (DP) technique (Eppstein et al 1992) that builds chains of fragments (local alignments).

2.1 Fragment Chaining

Representation of Fragments: For the purpose of this paper, each fragment is represented as a 7-tuple `(Start1, End1, Start2, End2, Score, Strand, CName)`. The first four numbers represent substrings of Sequence 1 and Sequence 2 respectively. `Score` is assigned to the fragment by the local alignment algorithm that produced it. `Strand` is either `+/-` , representing either a hit of the reference on the forward or reverse strand of sequence 2. Finally `CName` is an identifier indicating which contig of Sequence 2 the fragment belongs to.

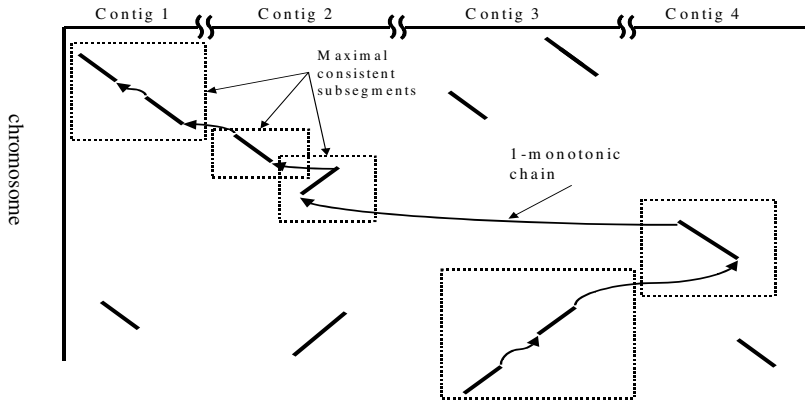

Fig. 1. Draft v/s Finished Alignment: Local alignments are chained together, series of fragments that belong to the same contig, and that are rearrangement free form a single syntenic block.

Scoring: Our scoring scheme is based on SLAGAN (Brudno et al 2003c). We find the highest scoring 1-monotonic chain, which is a subset of fragments that increase monotonically in Sequence 1 coordinates, but there is no requirement for monotonicity on the Sequence 2 axis. Relaxing this requirement allows us to capture rearrangements. We now describe a scoring scheme that scores chains, when the both sequences under consideration are finished sequences.

For a chain C, score(C) = Σ_i (F_i.score – $g_{case}(f_{i-1},f_i)$).

Here g_{case} is one of 8 penalty functions. The 8 functions correspond to 8 cases based on $(Strand_{i-1}, Strand_i, Transposition)$. Transposition is a boolean value that is `TRUE` if a transposition has occurred between the two fragments. Each of these functions is an affine expression on the L1 distance between the end of one fragment and the start of the next fragment, and the difference in the diagonals of the

fragments. For a comprehensive description of the scoring functions and the algorithm to find the best such chain refer to (Brudno et al 2003c). The algorithm runs in $O(nlgn)$, where n is the number of fragments.

Handling Contigs: We arrange the contigs in arbitrary order on the Sequence 2 axis. See Figure 1. We introduce a new scoring function $g_{unrelated}$, which is applied if the adjacent functions belong to different contigs. This function is a large constant penalty, though one could also use an affine function of the sequence 1 distance between the two fragments to penalize long stretches of sequence with no alignment.

We modify the original sparse DP algorithm to handle contigs on the Sequence 2 axis and the unrelated penalty without change in the asymptotic space or time requirements.

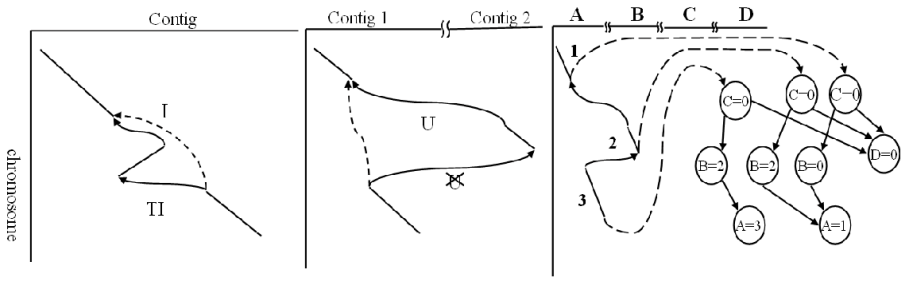

Fig. 2. 2a shows an inversion event being over penalized, 2b shows a contig splice event being penalized twice. 2c shows the structure of the overlap tree for three fragments (1,2,3) across two contigs (A,B).

2.2 Scoring Issues

In practice, the scoring scheme described above suffers from a few problems. Fragment chaining methods score transitions between adjacent fragments on a chain instead of scoring rearrangement events. Here are two cases where it affects the quality of the map produced.

CASE 1: We pay both an inversion penalty, and a translocated inversion penalty for a simple inversion (Figure 2a).
CASE 2: For local alignments from one contig spliced between local alignments from another contig, we pay the unrelated penalty twice (Figure 2b).

One possible model that does capture nested rearrangements explicitly is a stochastic pair-CFG. This is a generalization of a stochastic CFG (Eddy,Durbin 1994) that generates two sequences simultaneously. Though it is possible to write down a set of productions that capture all nested rearrangements, the task of finding the optimal parse is computationally expensive ($O(N^6)$ for two sequences of length N each.). To our knowledge, this formulation has not been proposed previously. One possible heu-

ristic is to work with fragments produced by a local aligner, rather than individual basepairs. We introduce a further heuristic that adds information to the chaining procedure, to make better chaining decisions.

The chaining procedure works in a top down manner. The sparse DP works by maintaining a set of active fragments, to which other fragments yet to be considered can chain. For each such active fragment F, we maintain a $\texttt{CINFO}_{\texttt{F}}$ query structure. Such a structure maintains a reference to the last fragment on the chain ending at F, for each value for the tuple (CName, Strand), if such a fragment exists. Here CName $\in$ **ContigNames** (which is the set of identifiers of all the contigs), Strand $\in$ {+,-}.

The scoring proceeds as follows: Fragment $\texttt{f}_{\texttt{Child}}$ $(\texttt{CName}_{\texttt{Child}}, \texttt{Strand}_{\texttt{Child}})$ chains to $\texttt{f}_{\texttt{Parent}}(\texttt{CName}_{\texttt{Parent}}, \texttt{Strand}_{\texttt{Parent}})$, we apply a penalty corresponding to the minimum of the following quantities:

1. $g_{case}(f_{Child}, CINFO_{fParent}(CName_{Child}, +))$
2. $g_{case}(f_{Child}, CINFO_{fParent}(CName_{Child}, -))$
3. $g_{unrelated}$.

The issues of the previous scoring scheme described in the section are resolved: the transition in CASE 1 is penalized as a normal transition instead of an inverted translocation as the third fragment is penalized against the first instead of the second, though it is chained to the second. Similarly, the transition in the CASE 2 is penalized as a normal transition instead of a contig transition event.

2.3 Application of OvBSTs to the Context Fragment Chaining

The following observation allows us to implement the CINFO structure efficiently: If fragments $\texttt{f}_{\texttt{Child}}$ chains to $\texttt{f}_{\texttt{Parent}}$, then the contents of $\texttt{CINFO}_{\texttt{Child}}$ differs from that of $\texttt{CINFO}_{\texttt{Parent}}$ only in the fragment corresponding to $(\texttt{CName}_{\texttt{Child}}, \texttt{Strand}_{\texttt{Child}})$. All other queries would return the same answer. We use a Overlapping Binary Search Tree (OvBST) (Burton and Huntbach 1985) to efficiently maintain the CINFO structure. An OvBST is a collection of binary search trees each with a distinct root, but share paths.

Let C be the number of contigs, n be the number of fragments.

DATA STRUCTURE: Each node of the OvBST has: key:= (CName, Strand); data:=(Reference to Fragment)

SETUP: Create statically a balanced binary search tree, over the key space (**ContigNameNames**, {+,-}). The time taken for this step is O(ClgC).

QUERY: A query to the CINFO structure of a fragment is a binary search starting at the root corresponding to the fragment. This operation is performed identically to a find key operation in a regular search tree. It takes time $O(\lg C)$.

CREATE: To create a CINFO structure for a fragment, given the CINFO structure for the parent. Let X be the node corresponding to the child's (CName, Strand) in the parent's CINFO structure. A copy is made of all the nodes on the path between X and the root of the parent. Note that all the pointers are copied as well. The data field in the node corresponding to the child is changed to refer to the child fragment. The operation takes time proportional to the height of the tree, which is $O(\lg C)$, and at most $O(\lg C)$ nodes are copied to make a CINFO structure for the child fragment.

The overlap tree modification allows the sparse DP algorithm to run in time $O(n \lg n \lg C)$, and space $O(n \lg C)$ making it practical for genome-sized datasets.

3 Whole Genome Alignment and Results

Using the draft fragment chaining algorithm described above we have generated a whole genome alignment between *S. cerevisiae* and each of *S. paradoxus, S. mikatae*, *S. kudriavzevii*, *S. bayanus*, *S. castellii* and *S. kluyveri* by repeating the following steps for each of the 16 chromosomes of *S. cerevisiae* (the finished sequence) and each of the other genomes (the draft contigs):

1. Generation of local alignments between the sequences using CHAOS (Brudno and Morgenstern 2002). We used an exact matching 12-mer as a minimal seed, and each local alignment after extension had to have a score of at least 2500 (see the LAGAN toolkit manual available from http://lagan.stanford.edu for an explanation of these and all other parameters).

2. Generation of the 1-monotonic conservation map using the standard SLAGAN penalties. We used an unrelated penalty of 15000 for the five yeast genomes, and a lower 7000 unrelated penalty for *S. kluyveri*, which is not only the most distant to *S. cerevisiae* but also has the poorest assembly.

3. Extension and alignment of all consistent subsegments in the 1-monotonic conservation map with the LAGAN aligner (Brudno et al. 2003b).

4. Glueing together the global alignments from step 3 to form a single global alignment between the chromosome and the draft contigs. Note that the expansion step may cause an overlap in the first sequence. This overlap is resolved by clipping the generated alignments. The optimal clipping point is found by a linear pass over the overlapping region. See (Brudno et al 2003c) for further explanation.

The results of the alignment are summarized in Tables 1 and 2. In constructing the statistics for each species we considered only the exons alignable by a protein-based local aligner: those that were covered more that 90% of their length by TBLASTX (Altschul et al 1997) alignments with a protein-level identity > 50%.

Table 1. Nucleotide Coverage Statistics: Table number of times nucleotides of each species was aligned. The numbers are in percentages.

	0	1	2	3	4	5	>5-10	>10-20	>20
S-paradoxu	4.24	90.67	3.94	0.84	0.15	0.04	0.04	0.05	0.05
S-mikatae	5.83	90.02	2.97	0.58	0.17	0.12	0.18	0.09	0.05
S-kudriavz	5.41	90.94	2.54	0.40	0.23	0.08	0.18	0.16	0.05
S-bayanus	4.80	91.93	2.24	0.63	0.16	0.04	0.07	0.06	0.07
S-castellii	16.45	65.08	15.61	2.18	0.40	0.17	0.04	0.03	0.04
S-kluyveri	29.50	42.88	24.17	2.39	0.57	0.14	0.19	0.09	0.07

Table 2. Exon Conservation Statistics: Table shows the percentage of exons that have a particular percent conservation rate. The second column denotes the size of the TBLASTX-pruned set of exons for each species, as explained above.

Exon Conservation Stats for 14820 exons

	# Exons	<45	45-55	55-65	65-75	75-85	85-95	95-100
S-paradoxus	**13674**	0.44	0.16	0.23	1.50	9.22	77.63	10.82
S-mikatae	**12620**	1.16	0.46	0.96	5.19	50.73	33.66	7.83
S-kudriavzevii	**11446**	0.96	0.67	1.65	10.76	57.60	20.33	8.04
S-bayanus	**11694**	0.84	0.62	2.25	14.80	60.32	13.77	7.41
S-castellii	**5004**	11.22	3.90	20.51	31.30	13.88	9.57	9.61
S-kluyveri	**4606**	16.44	6.56	20.16	24.09	13.07	10.63	9.09

For all of the sensu-stricto sequences, more than 90% of the nucleotides of the assembly were mapped to a single location in the yeast genome. An additional 5% was unmapped, while no more than 1.2% of any genome was mapped more than twice (see below for a discussion of the nucleotides mapped twice). These statistics attest to the specificity of the algorithm when comparing the more closely related sequence. Additionally, none of these four genomes had more than 2% of exons conserved at less than 55%, indicating that the resulting algorithm is sensitive enough to align the important coding elements.

For the two more distant genomes our algorithm maintained a low false positive rate (~3% of the genome mapped to more than two places), but the exon conservation dropped significantly: 15% of the alignable *S. cerevisiae* exons were conserved less than 55% with *S. castellii* and 23% with *S. kluyveri*. These numbers reflect the lower biological conservation between *S. cerevisiae* and these genomes, and the difficulty in aligning a distant, highly fragmented genome (the *S. kluyveri* was the worst quality genome with the shortest contigs; *S castelli*, the third worst).

It has recently been reported that the yeast genome has undergone whole genome duplication, followed by extensive loss of up to 90% of all genes through short deletions (Kellis et al. 2004). This result correlates with the large number of nucleotides that we have found mapped twice from *S. kluyveri* and *S. castellii* to *S. cerevisiae*. Because the genes are lost over time, the most distant sequences are more likely to lose different genes, forcing the alignment of some contigs in two places, once for each of the two copies.

4 Draft Genome Co-assembly

In this section we describe an algorithm to co-assemble two assemblies of the same genome, based on the sparse DP-chaining technique from above. The genome of *Ciona savignyi* has been sequenced to draft quality by a standard whole genome shotgun (WGS) strategy (*http://www.broad.mit.edu/annotation/ciona/*). WGS sequencing and assembly entails randomly breaking DNA from a genome into fixed-size pieces (inserts), and sequencing a 'read' from both ends of each insert (Fleischmann, Adams et al. 1995). Reads from opposite ends of a single insert are termed paired reads, and the distance between them on the sequence can be estimated by the size of the insert. An assembler program rebuilds the genomic sequence by combining reads with sequence overlaps into contigs. Contigs are organized into scaffolds by linking contigs using paired read information.

Like most multi-cellular organisms, *C. savignyi* individuals carry exactly two copies of every chromosome in their genome, which are referred to as the two haplotypes. One copy of each chromosome in an individual is inherited from each parent, and it is not possible to separate out individual chromosomes prior to WGS sequencing. Differences between copies of the same chromosome, called polymorphisms, can take the form of individual base pair substitutions or insertions and deletions that can range from a few base pairs to several thousand base pairs in length. It is these polymorphisms that make every individual within a species unique.

Polymorphism rates vary between species, and current WGS assembly programs (Arachne (Batzoglou et al. 2002, Jaffee et al. 2003), Phusion (Mullikin et al. 2003)) are not designed for highly polymorphic genomes: the polymorphism rate in the human genome is estimated to be only 0.1%. *C. savignyi*, on the other hand, has a very high rate of polymorphism, estimated to be more than 7%. When dealing with such a genome, the assembler (Arachne) places reads from the same position in the genome but from different haplotypes into separate contigs. The resultant assembly of *C. savignyi* thus contains two distinct copies of the genome, each of which is fractured. According to our observations, contigs and scaffolds contain several misassemblies because of the assembler's handling of the varying rate of polymorphism across the genome.

It is desirable to build a joint assembly of the two haplotype assemblies to each other for two primary reasons: (1) to identify regions of disagreement which highlight potential errors in the assembly, and (2) to provide a global alignment from which a *C. savignyi* single reference sequence can be built. Such a sequence can then be used to more accurately predict genes and other functional elements in the genome.

4.1 Strategy for Draft – Draft Alignment

Construction of a Bipartition: Initially the scaffolds are separated into two sets, each set corresponds to a single haplotype. Each scaffold is represented as a node, and there is an edge between two contigs if there is significant local similarity between them. In the absence of large-scale duplications the resulting graph is bipartite, and all the nodes in each partition come from a single haplotype.

Genome Co-assembly: The underlying idea behind the co-assembly of two haplotypes is that each haplotype can be used to establish an ordering of the contigs and scaffolds in the other haplotype. Each haplotype is now a set of contigs (contiguous stretches of DNA sequence). The contigs are ordered into scaffolds by assembly links. These assembly links are based on paired reads, and are less reliable than the contigs that they join. To order haplotype X we use all contigs of haplotype Y as the "chromosomes" for the sparse DP-chaining algorithm described above. Because the two genomes being co-assembled are very similar (these genomes are from the same individual), we use high thresholds for homology.

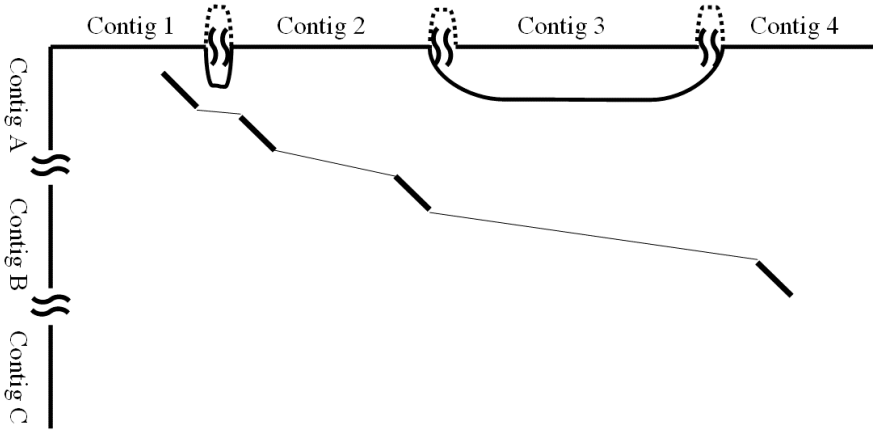

Fig. 3. Dotted connections between contigs are assembly links and solid connections are alignment links. Contigs 1 & 2 are joined by both as the alignment link (through Contig A) confirmed the assembly link. Contigs 2 & 4 were joined by an alignment link, causing the rejection of the assembly links from contig 3 to both contigs 2 and 4.

This step is summarized in Figure 3. Initially we set forward and backward links for each contig to its neighbors in the scaffold. Whenever two contigs 1 and 2 of X are aligned next to each other and against a single contig A of haplotype Y based on the chain from our sparse DP algorithm, 1 and 2 are said to be joined by an alignment link, and the forward and backward links of the two contigs are set to each other. Note that any contigs that lie between 1 and 2 can be separated out into a new scaffold: if the in-between contigs match any sequence, they will be aligned separately; if they do not match any sequence, we use the sequence from haplotype Y to fill the sequence gap (as between contigs 2 and 4 in Figure 3). If a contig has multiple forward or backward alignment links, it is labeled unreliable, as it could be a site of a misassembly on the contig level (or a biological rearrangement). All links to unreliable contigs are removed. All connected components of the link graph are now joined into a contiguous sequence and the process is repeated in order to get a relative ordering of the connected components. During this step, only the reliable "scaffolds" of haplotype Y are used as a basis for ordering all of the "scaffolds" of haplotype X.

4.2 Co-assembly Results

We applied the algorithm to nineteen genomic regions representing approximately 10% of the *C. savignyi* genome. The nineteen regions contain a total of 38.8 Mbp of sequence, and are comprised of 3,283 contigs arranged into 211 scaffolds. The algorithm above was used to order the contigs and scaffolds, and the results were analyzed manually for 1) incorrect ordering of scaffolds, 2) change of order between contigs within a scaffold, and 3) locations where one scaffold was inserted into another one.

For some regions the algorithm ordered the scaffolds in two or three groups and did not order the groups; however, the overall ordering was correct in all cases. In nineteen cases the ordering of the contigs within a scaffold that was reported by our algorithm differed from the ordering given by the assembler (3 regions had 3 rearrangements, 3 regions had 2, and 5 had 1). Of these 19 rearrangements, 18 were judged to be true positives, with one false positive. In 16 of the 18 true rearrangements the resulting sequence was correctly ordered with respect to the opposite haplotype, as determined by manual inspection of the resulting alignments. Additionally, in 42 instances an entire scaffold was inserted into the middle of another scaffold, and all 42 of these cases were detected and ordered correctly. A full analysis of the co-assembly of the genome of *C. savignyi* will be published separately (K. Small, A. Sidow, et al., manuscript in preparation).

5 Conclusions

One of the main requirements of sequence comparison algorithms is the ability to process megabase-long sequences in a reasonable amount of time. While there has been extensive work in developing effective local and global alignment algorithms, these methods have commonly been too inaccurate to align draft genomic sequences. Chaining is an attractive technique for alignment because it provides an $O(n \lg n)$ (n is the number of fragments) method for building larger alignments from short similarities. The technique is also applicable for multiple alignment (Abouelhoda and Ohlebusch 2003), as each additional sequence only requires an additional factor of $O(\lg n)$, while in most other alignment methods the running time is exponential in the number of sequences. Recent work has shown chaining to be one of the most effective ways of speeding up global alignment and reducing the false positive rate of local alignment. Additionally sparse DP has been successfully used to detect and classify rearrangements and to compare genome assemblies. In the current work we apply chaining techniques to alignment of draft (incomplete) genomic sequence, and show that it can be used both to align draft contigs against a finished chromosome and to co-assemble two sets of draft contigs in order to build a more accurate assembly of a genome based on two independent assemblies.

One shortcoming of chaining (and all other sparse DP) methods is that they are memoryless: the decision about the chaining of a particular fragment can only depend on the score of the chain leading up to it and not on the previous history. While it is possible to retrace the chain in order to find the relevant previous information (such as whether some piece of a sequence was already aligned) this would add an extra factor of $O(n)$ to the run time, making the chaining approach much less practical. In this paper we introduce a method to add state to the sparse chaining methods while suffer-

ing an `O(lg C)` runtime hit, where `C` is the number of "events" that is necessary to remember. This state information helps the fragment chaining algorithm to properly charge penalties for the various events that happen along the chain, and similar approaches can be useful in other algorithms that use sparse dynamic programming.

References

Abouelhoda, M.I., Ohlebusch, E. 2003. A Local Chaining Algorithm and Its Applications in Comparative Genomics. WABI, 1-16.

Altschul, SF, Madden TL, Schäffer AA, Zhang J, Zhang Z, Miller W, and Lipman DJ. 1997 Gapped BLAST and PSI-BLAST: a new generation of protein database search. Nucleic Acids Res 25(17):3389-3402

Batzoglou S, Jaffe D, Stanley K, Butler J, Gnerre S, Mauceli E, Berger B, Mesirov JP, Lander ES, 2002 ARACHNE: A whole genome shotgun assembler. *Genome Research* 12:177-189,

Bray, N., Dubchak, I., Pachter, L. 2003. AVID: A Global Alignment Program.Genome Research, 13:97-102.

Brudno, M., Chapman, M., Gottgens, B., Batzoglou, S., and Morgenstern, B. 2003a. Fast and sensitive multiple alignment of large genomic sequences. BMC Bioinformatics, 4(1):66.

Brudno, M., Do, CB, Cooper, GM, Kim, MF, Davydov, E, Green, ED, Sidow, A, and Batzoglou, S. 2003b. LAGAN and Multi-LAGAN: Efficient Tools for Large-Scale Multiple Alignment of Genomic DNA. Genome Research, 13(4): 721-731.

Brudno M., Malde S., Poliakov A., Do C.B., Couronne O., Dubchak I., Batzoglou S. 2003c. Glocal alignment: finding rearrangements during alignment. Bioinformatics. 19 Suppl 1:i54-62.

Brudno M, Morgenstern B. Fast and sensitive alignment of large genomic sequences. Proceedings of the IEEE Computer Society Bioinformatics Conference (CSB) 2002.

Burton, FW, Huntbach, MM. 1985. Multiple Generation Text Files Using Overlapping Tree. The Computer Journal, 28(4):414-416

Cliften P, Sudarsanam P, Desikan A, Fulton L, Fulton B, Majors J, Waterston R, Cohen BA., and Johnston M. 2003. Finding functional features in *Saccharomyces* Genomes by phylogenetic footprinting. Science, 301:71-76

Delcher AL, Kasif S, Fleischmann RD, Peterson J, White O, and Salzberg SL 1999. Alignment of Whole Genomes. Nucleic Acids Research, 27:11, 2369-2376

Delcher AL, Phillippy A, Carlton J, and Salzberg SL 2002. Fast Algorithms for Large-scale Genome Alignment and Comparision., Nucleic Acids Research , Vol. 30, No. 11 2478-2483.

Eddy SR and Durbin R. 1994 RNA sequence analysis using covariance models. Nucl Acids Res. 22:2079-2088,

Eppstein, D., Galil, R., Giancarlo, R., and Italiano, G.F. 1992. Sparse dynamic programming I: linear cost functions. J. ACM, 39:519-545.

Fleischmann, RD, Adams, MD, White, O, Clayton, RA, Kirkness, EF, Kerlavage, AR, Bult, CJ, Tomb, JF, Dougherty, BA, Merrick, JM, et al. 1995. Whole-genome random sequencing and assembly of Haemophilus influenzae. Science, 269(5223):496-512.

Jaffe DB, Butler J, Gnerre S, Mauceli E, Lindblad-Toh K, Mesirov JP, Zody MC, and Lander ES. 2003. Whole-genome sequence assembly for mammalian genomes: Arachne 2. Genome Res. 13(1):91-6.

Kellis, M., Birren, B., Lander, ES. 2004. Proof and evolutionary analysis of ancient genome duplication in the yeast Saccharomyces cerevisiae. Nature, 428:617-624.

Kellis, M., Patterson, N., Endrizzi, M., Birren, B., Lander, ES. 2003. Sequencing and comparison of yeast species to identify genes and regulatory elements. Nature, 423:241-54.

Lippert, R.A., Zhao, X., Florea, L., Mobarry, C., and Istrail, S. 2004. Finding Anchors for Genomic Sequence Comparison. Proceedings of ACM RECOMB 2004.

Mullikin, J.C., Ning, Z. 2003. The phusion assembler. Genome Res, 13(1):81-90.
Needleman, SB. and Wunsch, CD. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. 48, 443-453.
Smith, TF and Waterman, MS. 1981. Identification of common molecular subsequences. J. Mol. Biol. 147, 195-197.
Tzouramanis, T., Vassilakopoulos, M., Manolopoulos, Y. 2000. Multiversion Linear Quadtree for Spatio-Temporal Data. DASFAA.
Veeramachaneni, V., Berman, P., Miller, W. 2003. Aligning two fragmented sequences. Discrete Applied Mathematics, 127(1):119-143.