The Pennsylvania State University

The Graduate School

Department of Computer Science and Engineering

ALIGNING FRAGMENTED SEQUENCES

A Thesis in

Computer Science and Engineering

by

Vamshi Veeramachaneni

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2002

We approve the thesis of Vamshi Veeramachaneni.

Date of Signature

Piotr Berman
Associate Professor of Computer Science and Engineering
Thesis Adviser
Chair of Committee

Webb Miller
Professor of Computer Science and Engineering

Jonathan Goldstine
Associate Professor of Computer Science and Engineering

Wojciech Makalowski
Associate Professor of Biology

Raj Acharya
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering

Abstract

The past few years have seen a remarkable series of accomplishments related to DNA sequencing, with genome sequences being generated for several key organisms including a yeast, a nematode, a plant and the human. Upon completion of the human and mouse genome sequences, world-wide sequencing capacity will turn to other complex organisms. Because of the prohibitive cost of fully sequencing a genome, current strategies call for many of these genomes to be incompletely sequenced. That is, gaps will remain in the known sequence, and the relative order and orientation of the known sequence fragments may not be determined.

Sequence comparison between two genomes of this sort may allow some of the fragments to be oriented and ordered relative to each other by computational means. We formalize this as an optimization problem, show that the problem is MAX-SNP hard, and develop a polynomial time algorithm that is guaranteed to produce a solution whose score is within a factor 3 of optimal.

We also consider the simpler problem where only one of the genomes is fully sequenced. We describe a heuristic that generates near optimal solutions on real data sets and show that solutions with high score have correspondingly high biological significance.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank my advisor, Piotr Berman, for his patience and constant encouragement. I will always treasure the numerous hours we spent in random conversations. His boundless curiosity and analytical approach to every single topic have been a source of inspiration during my stay at Penn State. I am also grateful to Webb Miller for introducing me to the wonderful world of computational biology.

I am forever indebted to my friends and former roommates – Aniruddha Vaidya and Shailabh Nagar. If not for them, I would not have joined the Association for India's Development (AID) and would likely still be a bachelor. I am also grateful to my friends of twenty odd years, Appa Rao and Kishore Reddy, for staying in touch all through my highs and lows.

My parents are responsible for my academic achievements in more ways than I can count. Their encouragement and passion for learning have inspired me and my sisters all through our careers.

Finally, I am grateful to my wife, Kavitha, for magically bringing my life together.

Chapter 1

Introduction

The past few years have seen remarkable progress in the field of DNA sequencing. Some of the landmark achievements are the sequencing of several key experimental organisms including a yeast (*Saccharomyces cerevisiae*) [19], the multicellular organism *Caenorhabditis elegans* [13], the plant *Arabidopsis thaliana* [22], the fruitfly *Drosophila melanogaster* [1, 28] and the completion of the draft sequence of the human genome [46, 12]. At present, the complete genome sequences of 91 species (mostly bacteria) are available in the Genomes section of GenBank.

The GenBank sequence database [5], a repository of all publicly available DNA sequences, continues to grow at an exponential rate, doubling in size approximately every 14 months. As of release 130, June 2002, GenBank contained over 20.64 billion nucleotide bases from 17.47 million different sequences. Complete genomes represent a growing portion of the database, with 62 of the 84 complete microbial genomes now in GenBank deposited over the past 3 years.

However, current costs associated with large-scale sequencing remain a limiting factor for genome sequencing efforts [20]. For example, establishing approximately one-fold sequence coverage of a mammalian genome requires the generation of roughly 6 million sequence reads and at present costs about US $10–20 million (and so, a working draft sequence with approximately five-fold coverage would cost US $50–100 million). As a result, even though more than 100,000 different organisms are represented in GenBank, it is clear that only a handful of different genomes can realistically be sequenced in a comprehensive fashion, at least by the available methods.

The sequencing strategies that we describe in Chapter 2, do not sequence genomes completely. Instead, they produce a sequence with gaps, i.e., consisting of many fragments with undetermined order and orientation. In this thesis, we show that some order and orientation relationships can be deduced by comparing two fragmented genomic sequences.

The underlying assumption in our work is that genomic organization tends to be evolutionarily conserved between species. This idea was first postulated in the early 1900s [9, 21]. Since then studies have shown that large stretches of DNA are conserved in mammalian species as divergent as human and fin whales [29, 41, 42, 48]. Other studies evaluating genomic conservation at the genetic and physical mapping levels have determined that gene order does tend to be conserved among mammals [31, 44]. We describe various chromosomal rearrangements and the concept of conserved segments in Chapter 2. We also discuss in brief recent works in sequencing, annotation and comparative map generation that are based on the presence of large conserved segments.

In Chapter 3, we formalize our optimization problem. We show that a simple version of the problem is MAX-SNP hard and we develop a polynomial time algorithm that is guaranteed to produce a solution whose score is within a factor 3 of the optimal. The formal developments presented in this chapter, including results showing how algorithms for certain simpler problems can be combined to solve a more general problem, provide a conceptual framework for designing effective heuristics.

In Chapter 4, we consider the simpler problem in which one of the genomes is assembled but the other is in unordered fragments. We describe a heuristic that uses some of the techniques developed in Chapter 3 and we evaluate its performance on real data sets. For our test data we show that the heuristic generates solutions that are within 5% of the optimum and that solutions with high scores have correspondingly high biological significance.

Chapter 2

Background

2.1 Sequence Assembly

2.1.1 Shotgun sequencing

First described in the early 1980s [18, 2, 39, 15, 26], the shotgun sequencing method shown in Figure 2.1 is used in all large scale sequencing techniques. In this method, multiple copies of the DNA fragment to be sequenced are created by cloning and then randomly partitioned into smaller fragments. Approximately 500 consecutive basepairs from the ends of these fragments can be determined using variations of the Sanger method [40]. These sequences are called *reads*. Overlapping *reads* are assembled into *contigs*, i.e., presumably contiguous sections of the genomic sequence. The ideal outcome of this step is a single contig that represents the entire sequence of the original DNA fragment. More commonly, the result is a set of contigs with unknown order and orientation (strand information).

For example, consider the DNA fragment shown in Figure 2.2. Each of the strands is represented by a linear arrangement of four different bases—adenine(A), thymine(T), guanine(G) and cytosine(C). The opposite strands are linked by hydrogen bonds between the guanine and cytosine and between the adenine and thymine. Since strands are read in the $5' \rightarrow 3'$ direction, the fragment is represented by the sequence $s = \texttt{AAAAG}\cdots\texttt{GTGG}$ or its reverse complement $s^R = \texttt{CCAC}\cdots\texttt{CTTTT}$. Suppose the result of shotgun sequencing is the set of contigs $\{f_1, f_2, f_3, f_4, f_5\}$. Our goal is to order the contigs along one of the strands. We call this the order/orient problem. The solutions in this instance are $\langle f_1, f_4^R, f_2, f_5^R, f_3\rangle$ and $\langle f_3^R, f_5, f_2^R, f_4, f_1^R\rangle$.

We now describe commonly used sequencing techniques and how they solve the order/orient problem.

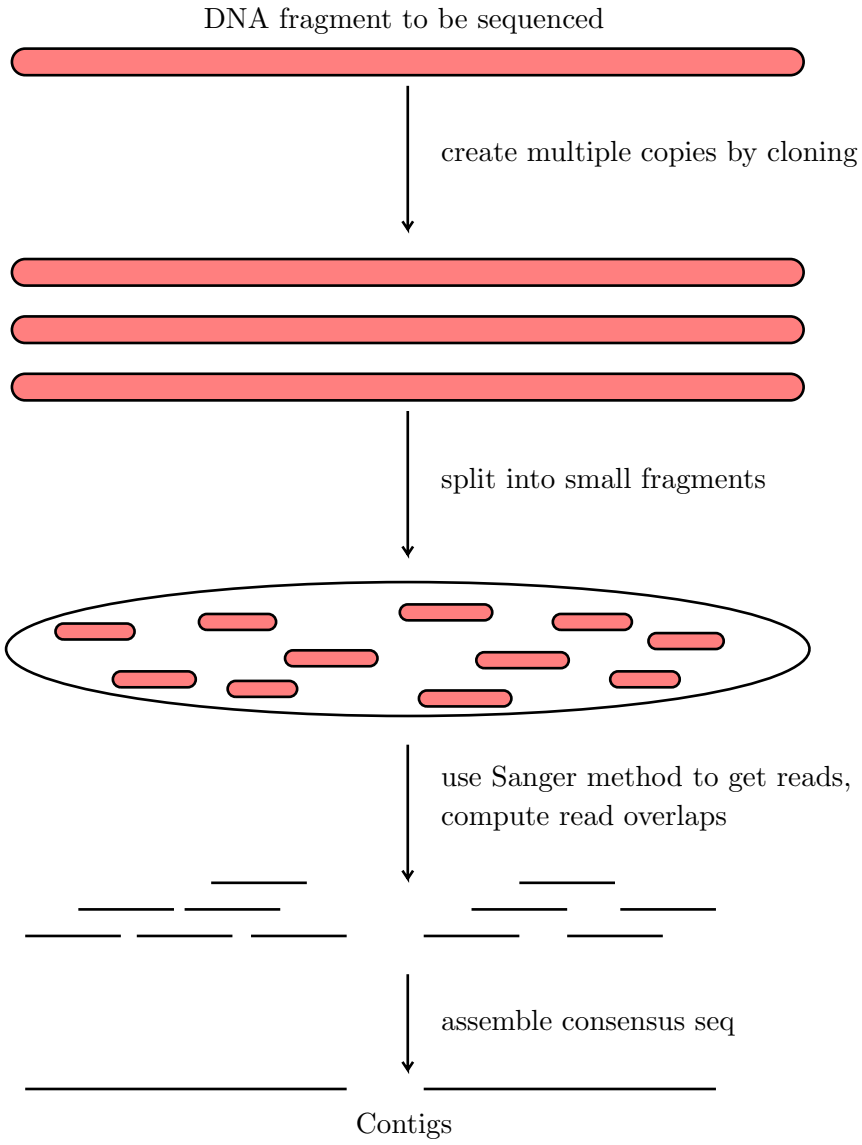

Fig. 2.1. Shotgun sequencing

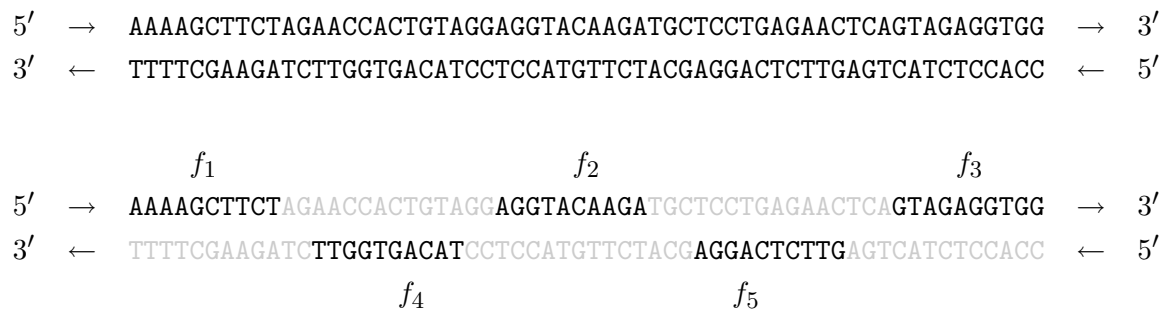

Fig. 2.2. A sample order/orient problem instance. Sequencing the DNA fragment shown in the upper half of the figure results in five contigs $f_1, \ldots, f_5$ with unknown order and orientation. The figure in the lower half shows the locations of the contigs in the original fragment

2.1.2 Clone-by-clone shotgun sequencing

The clone-by-clone method [10] adopted by the Human Genome Project (HGP) is the most commonly used approach for determining the sequence of large, complex genomes. It is also referred to as hierarchical shotgun sequencing or map-based shotgun sequencing. The major steps in this method are the following.

map construction a large number of pieces (of size $\sim$ 200 Kbp) of the target DNA fragment are cloned. This collection of clones, called a *library*, has no obvious order indicating the original position of the cloned pieces on the chromosome. Overlaps among clones in the library are detected using restriction enzyme based fingerprint analysis [33, 38, 14] or by the shared presence of unique DNA landmarks. These overlaps are used to assemble the clones into a *clone map* covering the entire target DNA.

clone selection minimally overlapping clones that form a minimal tiling path across the clone map are selected. This set of clones is called a *sequence ready clone map*.

shotgun phase individual clones of the sequence ready clone map are shotgun sequenced. The clone map imposes a partial order on the sets of contigs obtained. Finally, the contigs of each clone are ordered and oriented using a high resolution map of markers.

The clone map construction step essentially decomposes a large genome into several smaller genomes and allows for a more efficient organization of distributed resources and sequencing capacities. Assemblies generated with even 2–5-fold coverage can be used in many analyses [8].

2.1.3 Whole genome shotgun sequencing (WGS)

The WGS approach [47] skips the clone map construction step and directly sequences a large number of unmapped clones. For further assembly it uses a library of pairs of reads, called *mates*, from the ends of long inserts randomly sampled from the genome. Since the lengths of these long inserts can be measured accurately, the presence of mates in different contigs serves to order and orient the contigs and give the approximate distance between them. The result of this assembly process is, therefore, a collection of *scaffolds*, where each scaffold is a set of contigs that are ordered, oriented and positioned with respect to each other.

In clone-by-clone based methods, sequence assembly programs only encounter data sets derived from 100–200 kb clones. WGS assemblers, on the other hand, need to be much more robust and sophisticated since they deal with data sets that are orders of magnitude larger and include algorithms that must account for the anticipated spatial relationship of mates emanating from different clones.

2.1.4 Our method

In the absence of high resolution marker maps or mate pair information, we can still infer some order/orient relationships by comparing the conserved regions present in contigs of two organisms that are close in evolutionary terms. This process was performed manually by Onyango *et al.* [34]. Figure 2.3 illustrates the sort of inference that is possible.

Clearly, this method works best if a large number of conserved regions are present in the same order in both the genomes. In the next section, we describe briefly the biological processes underlying genome sequence conservation.

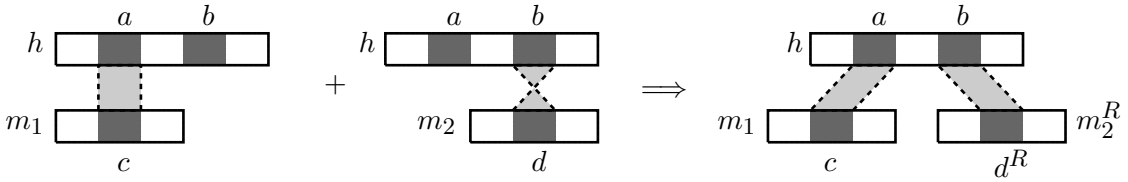

Fig. 2.3. Use of sequence comparison to orient/order contigs. Contig h (say, of human) includes region a, which aligns with region c in contig m_1 (say, of mouse). Also, another region of h, denoted b, aligns with d^R, the reverse complement of region d of mouse contig m_2. We infer that m_1 precedes m_2^R, relative to the orientation in which h is given. Note that the distance between m_1 and m_2^R *cannot* be inferred from such comparisons.

2.2 Conserved segments in chromosomes

During evolution, inter- and intra- chromosomal rearrangements such as fission, fusion, reciprocal translocation, transposition and inversion disrupt the order of genes along the chromosome, as depicted in Figure 2.4.

However, gene order remains fixed in the segment between any two neighboring breakpoints resulting from the rearrangements. These *conserved segments* tend to become shorter with time as they are disrupted by new events, and by the same token, the number of segments increases with each disruption. Comparative mapping studies have shown that closely related organisms share large conserved chromosome segments while more distantly related species exhibit shorter conserved segments [36, 17]. A high degree of genomic conservation is the predominant mode for the mammalian genome, for instance, the cat genome can be reorganized into the human genome by as few as 13 translocation steps [32]. However, exceptions to the slow rate of rearrangements occur in several mammalian orders. Rodent species, particularly mouse and rat, show rapid pattern of change, with some 180 conserved segments shared between human and mouse and 60 shared between mouse and rat.

We now review some applications of long conserved segments.

mapping A gene map shows the order in which genes are present on the different chromosomes.

For many species the number of known genes is much higher than the number of mapped

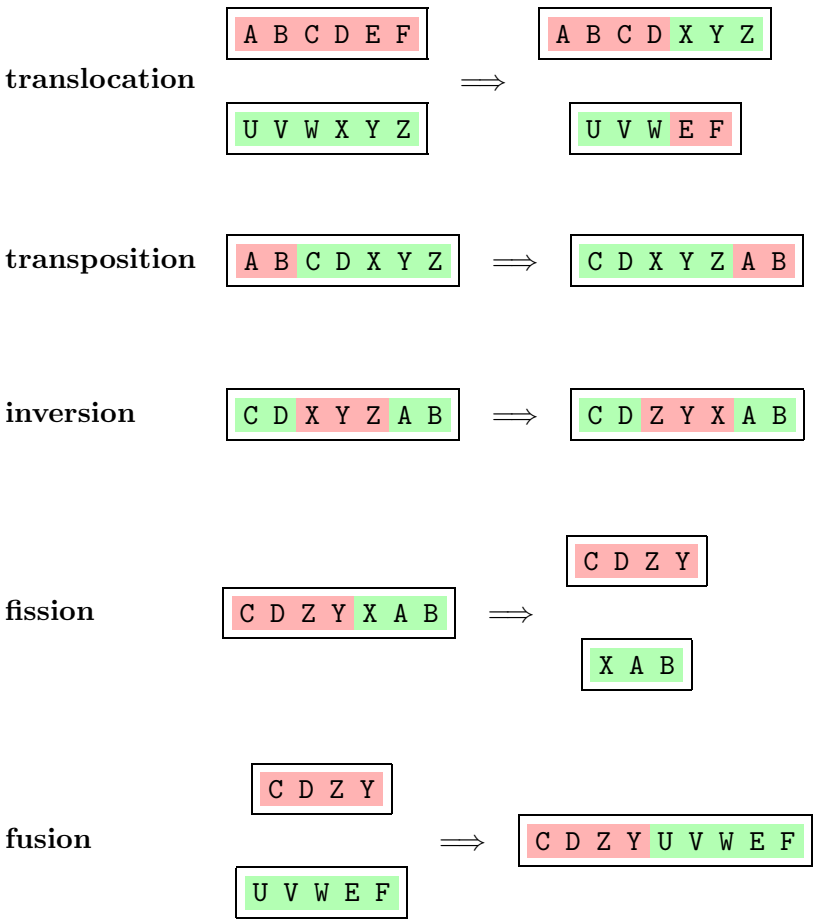

Fig. 2.4. Chromosomal rearrangments. Colors are used to highlight the changes in gene order. Assuming that the rearrangements take place in the order shown, the final chromosomal segment contains 4 conserved segments—(C D), (Z Y), (U V W) and (E F)—from the initial chromosomes.

genes. Comparative mapping, as shown in Figure 2.5, is currently being used [30, 25] to predict the position of unmapped genes in rat by inference from known positions in human and mouse.

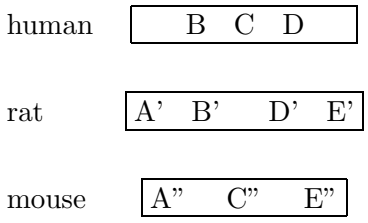

Fig. 2.5. Mapping a gene by inference. Genes with the same letters are orthologs. Let C' be an unmapped gene in rat. By comparing the rat, mouse gene maps we infer that C' should be placed between A' and E'. By using the human gene map we refine the prediction and place C' between B' and D'.

sequencing In clone-by-clone sequencing, a map of markers is used in the clone map construction step. A dense map of markers is helpful in identifying small overlaps among clones in the library, and by extension in the construction of sequence ready clone maps with small number of clones. Maps of markers for a particular section of the human genome can be used to create dense maps for homologous mouse regions. These maps have been shown to be effective in the construction of sequence ready clone maps of high quality [45, 24]. Maps created using this approach have also been used for ordering and orienting mouse contigs assembled from low-pass (2.2x coverage) shotgun sequencing [37].

annotation New gene predictions can be made based on sequence conservation at corresponding positions in conserved segments especially if the sequence matches some ESTs or is predicted by a gene prediction program [37].

Chapter 3

Consensus Sequence Reconstruction (CSR) Problem

We model the problem of determining order/orient relationships from alignments between contigs as follows. Data consists of a set of "h-contigs" and a set of "m-contigs", where each contig is simply an ordered list of conserved regions having associated alignment scores. We use $\sigma(a, b)$ to denote the score of the alignment between a and b, where a or a^R is a conserved region of an h-contig and b or b^R is a conserved region of an m-contig. An example of a permissible dataset consists of contigs $h_1 : \langle a, b, c\rangle$, $h_2 : \langle d\rangle$, $m_1 : \langle s, t\rangle$, $m_2 : \langle u, v\rangle$ and the alignment scores $\sigma(a, s) = 4$, $\sigma(a, t) = 1$, $\sigma(b, t^R) = 3$, $\sigma(c, u) = 5$, $\sigma(d, t) = \sigma(d, v^R) = 2$. See Fig. 3.1.

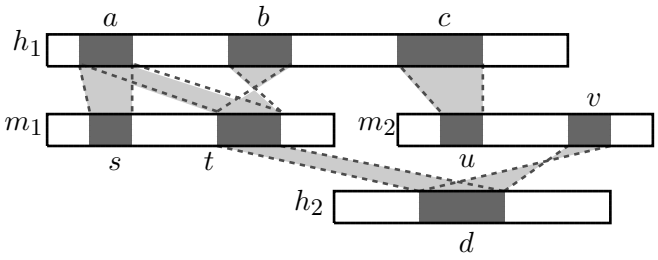

Fig. 3.1. Picture of a sample set of data, discussed in the text.

Alignments involving conserved regions in contig h_1 may serve to orient and order several m-contigs relative to each other. Some of these m-contigs may in turn orient and order h_1 relative to additional h-contigs, and so on. This leads to an "island"[1] of contigs that are oriented and ordered relative to one another. With ideal data, this process would partition the set of contigs into

[1]While similar to scaffolds [47], islands present a different combinatorial problem because they involve fragments of different species, do not imply any distance information and cannot overlap with other islands

islands, such that inter-island order/orient relationships cannot be determined from the alignments. In reality, the set of given alignments is frequently inconsistent with any proposed orientation and ordering of the contigs. Simple examples are shown in Fig. 3.2. More complex examples arise in practice when regions have been shuffled by evolutionary processes, when incorrect alignments are computed, and when contigs are incorrectly assembled from reads.

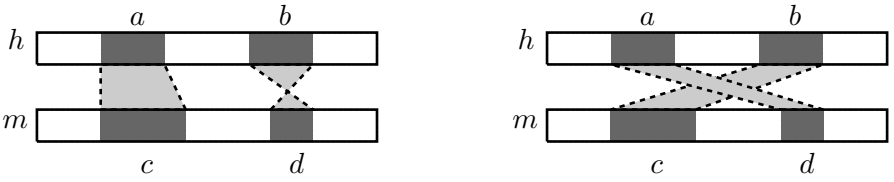

Fig. 3.2. Two potential inconsistencies among alignments between contigs. In the first example, contig h contains regions a and b, where a aligns with region c of contig m, and b aligns with d^R, where d is another region of m. The $a - c$ alignment supports the current orientation, while the $b - d$ alignment calls for reversal of m. The second example violates our requirement that aligning regions be in the same order in the two sequences.

Our goal is to determine orientations and an order for each of the two sets of contigs that, possibly together with deletions of some of the conserved regions, gives two equal-length and consistently ordered lists of conserved regions showing high overall similarity. Ideally, this would mean maximizing the sum of the scores σ. For a simple example, consider the dataset given several paragraphs above. We can delete (i.e., ignore) b and t, reverse h_2 and place it after h_1 (giving $\langle a, c, d^R \rangle$), then place m_1 before m_2 in their given orientation (giving $\langle s, u, v \rangle$), which yields the score $\sigma(a, s) + \sigma(c, u) + \sigma(d^R, v) = 4 + 5 + 2 = 11$. See Fig. 3.3 for a picture of the solution.

Note that once orientations and an order of the contigs are chosen, it is easy to decide how sites should be deleted to maximize the score—this is simply the classic problem of aligning two lists of symbols. (Here, however, each symbol of the "sequence" denotes a conserved region, rather

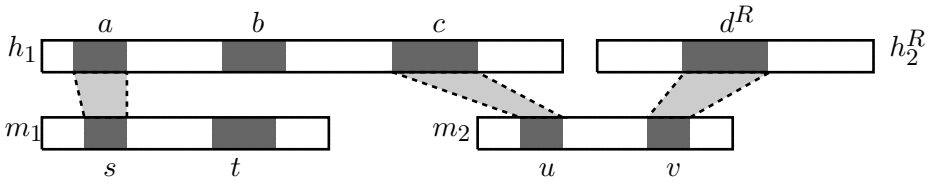

Fig. 3.3. Solution to the orient/order problem of Fig. 3.1. All alignments pictured in Fig. 3.1 that are inconsistent with this layout have been discarded.

than an individual nucleotide.) The difficulty lies with determining an optimal set of orient/order operations.

One of our results indicates that no polynomial-time algorithm can be guaranteed to orient and order the contigs so as to always maximize the resulting score. We show in this chapter that even if we make a number of simplifying assumptions, such as

- each conserved region is involved in precisely one alignment (e.g., for each a, $\sigma(a, b) > 0$ for just one b),
- there is only one m-contig and
- each h-contig has only two conserved regions,

the problem of computing an optimal set of orient/order operations is MAX-SNP hard (Theorem 3.2).

This formal result indicates that there exists a number $\alpha < 1$ such that any polynomial-time orient/order algorithm will sometimes produce a solution whose total score is less than α times the optimal score unless $P = \text{NP}$.

We develop a $(3 + \epsilon)$ approximation algorithm (Theorem 3.6) for the order/orient problem in Section 3.3. Our results showing how algorithms for certain simpler problems can be combined to solve a more general problem, provide a conceptual framework for designing effective algorithms for computing high-scoring orient/order operations.

3.1 Problem statement with variations

3.1.1 Consensus Sequence Reconstruction — CSR

Assume that we have two sets of DNA *fragments*, one for each species. Let us call these sets $\mathcal{H}$ and $\mathcal{M}$. We view each fragment as a sequence of *regions*. An *occurrence* of a region in a sequence can be *normal* or *reversed*.

Formally, we view each region as a symbol of a duplicated alphabet $\widetilde{\Sigma} = \Sigma \cup \Sigma^R$, and each fragment as a word from $\widetilde{\Sigma}^*$. To clarify the meaning of the reversal operation, we list its properties:

- $\Sigma \cap \Sigma^R = \varnothing$;
- for $a \in \Sigma$, $a^R \in \Sigma^R$, and for $a \in \Sigma^R$, $a^R \in \Sigma$;
- for $u, v \in \widetilde{\Sigma}^*$, $(uv)^R = v^R u^R$;
- for $u \in \widetilde{\Sigma}^*$, $(u^R)^R = u$;
- for $a, b \in \widetilde{\Sigma}$, $\sigma(a, b) = \sigma(a^R, b^R)$, where σ is a function $\sigma : \widetilde{\Sigma} \times \widetilde{\Sigma} \to \mathbf{R}$.

We introduce an extra padding symbol $\perp$ such that $\perp^R = \perp$ and we extend the score function σ by setting

$$\sigma(a, \perp) = \sigma(\perp, a) = 0, \quad \forall a \in \widetilde{\Sigma}$$

For $s \in \widetilde{\Sigma}^*$ we define the *set of padded sequences* $\mathcal{P}s$ as the set of sequences obtained from s by inserting the padding symbol $\perp$ an arbitrary number of times.

For $s, t \in (\widetilde{\Sigma} \cup \{\perp\})^*$, where $s = a_1 a_2 \dots a_l$ and $t = b_1 b_2 \dots b_{l'}$, we define

$$\text{Score}(s, t) = \begin{cases} 0 & \text{if } l \neq l' \\ \sum_{i=1}^{l} \sigma(a_i, b_i) & \text{otherwise} \end{cases}$$

Our general goal is to find an optimal conjecture for a consensus sequence for $\mathcal{H}$ and $\mathcal{M}$. More formally,

Definition 3.1. *For a set of fragments, $\mathcal{F} = \{f_1, \dots, f_k\}$, we define Conj($\mathcal{F}$), the set of valid conjecture sequences. A conjecture $\mathbf{f} \in$ Conj($\mathcal{F}$) is formed in three stages.*

1. *For each fragment f_i we select some padded sequence $s_i \in \mathcal{P}f_i$,*
2. *some of s_i's are replaced by their reversals,*

3. $\mathbf{f} = s_{\pi(1)} \dots s_{\pi(k)}$, *for some permutation* π *of* $[1, k]$.

A *conjecture pair* is $(\mathbf{h}, \mathbf{m}) \in \text{Conj}(\mathcal{H}) \times \text{Conj}(\mathcal{M})$. Our goal is to maximize $\text{Score}(\mathbf{h}, \mathbf{m})$.

3.1.2 Consistent match sets

Our algorithm will build conjecture pairs from smaller parts called *matches*, which pair together intervals selected from fragments of $\mathcal{H}$ and $\mathcal{M}$.

Given a fragment $h = a_1 \dots a_n$, the *site* $h(i, j)$ represents the contiguous subfragment $a_i \dots a_j$. A *match* is a pair of sites from fragments of different species.

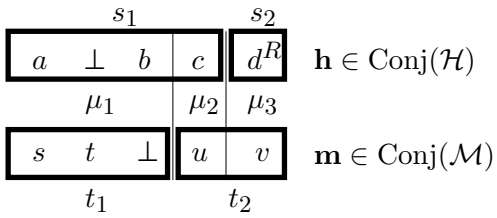

Fig. 3.4. The conjecture pair representing the solution shown in Fig. 3.3. Thick lines indicate padded fragments (or their reversals), thin lines separate matches. The set of matches consistent with this pair is $\mu_1 = (h_1(1,2), m_1(1,2))$, $\mu_2 = (h_1(3,3), m_2(1,1))$ and $\mu_3 = (h_2^R(1,1), m_2(2,2))$.

DEFINITION 3.2. *A conjecture pair* $(\mathbf{h}, \mathbf{m})$ *with a positive score produces a set of matches as follows:*

1. *Suppose* $\mathbf{h}$ *is formed as* $s_1 s_2 \dots s_k$ *and* $\mathbf{m}$ *is formed as* $t_1 t_2 \dots t_l$ *in Step 3 of Definition 3.1. We view this pair as a single word* $\mathbf{w}$ *where letters are columns of two symbols of* $\widetilde{\Sigma} \cup \{\perp\}$.
2. *As shown in Fig. 3.4, we split* $\mathbf{w}$ *at ends of* s_i*'s and* t_i*'s. The resulting pieces can be called* padded matches.
3. *Given a padded match, we obtain a match (pair of sites) by splitting it into two rows and deleting all* $\perp$*s from both the rows.*

A set of matches is consistent *if it is produced from some conjecture pair.*

DEFINITION 3.3. *If $h = h(1, n)$ then the sites in h can be classified as:*

full*:* $h(1, n)$ *or, equivalently,* h
border*:* $h(1, i)$ *or* $h(i, n)$
inner*: none of the above*

A match that involves a full site is called a *full match*, a match that involves a border site is called a *border match*. In Fig. 3.5 $\mu_1, \mu_4, \mu_5, \mu_7$ are full matches, and μ_2, μ_3, μ_6 are border matches. One can see that we need to consider these two kinds of matches only. Note that a match is a full match exactly when both ends of the respective padded match correspond to the ends of the same padded sequence, e.g., sequence t_3 in Fig. 3.5.

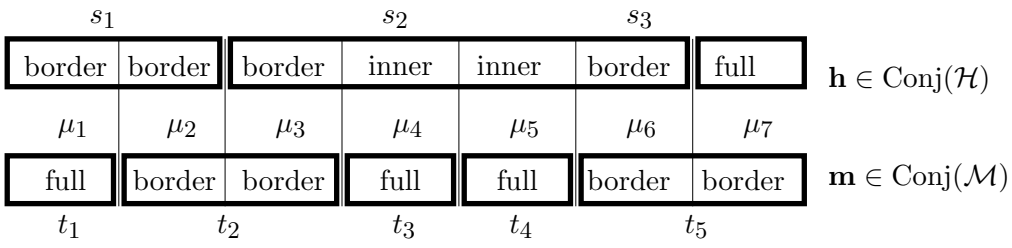

Fig. 3.5. A conjecture pair $(\mathbf{h}, \mathbf{m})$ divided into matches and the sites classified according to Definition 3.3.

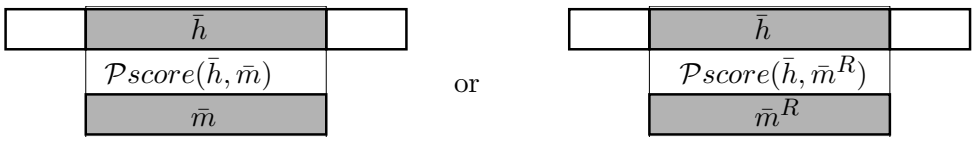

Fig. 3.6. Matches formed from an inner site and a full site.

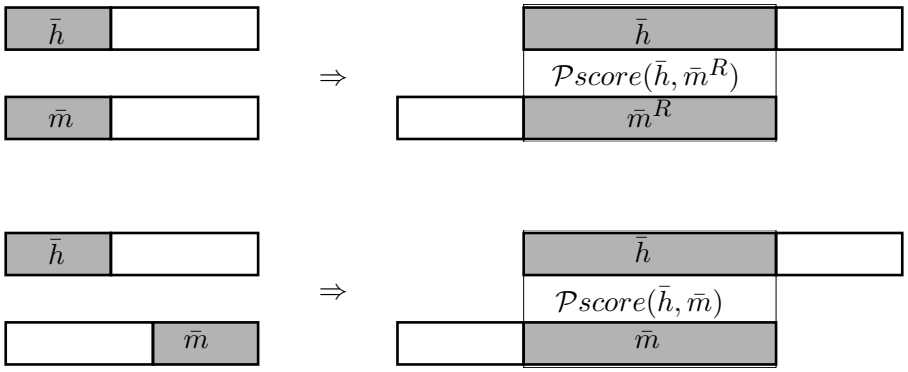

Fig. 3.7. Matches formed from border sites.

DEFINITION 3.4. *Given a site $\bar{h}$ in some fragment of $\mathcal{H}$ and a site $\bar{m}$ in some fragment of $\mathcal{M}$, we formulate the definition for* match score *$MS(\bar{h}, \bar{m})$ in several steps.*

- *For $\bar{h}, \bar{m} \in \widetilde{\Sigma}^*$,*

$$\mathcal{P}score(\bar{h}, \bar{m}) = \max_{u \in \mathcal{P}\bar{h}} \max_{v \in \mathcal{P}\bar{m}} Score(u, v)$$

- *If one of the sites $\bar{h}$, $\bar{m}$ is full then the match score of $\bar{h}$ and $\bar{m}$ is*

$$MS(\bar{h}, \bar{m}) = \max(\mathcal{P}score(\bar{h}, \bar{m}), \mathcal{P}score(\bar{h}, \bar{m}^R)) \qquad \textit{(See Fig. 3.6)}$$

- *If neither $\bar{h}$ nor $\bar{m}$ is a full site, then the match score of $\bar{h}$ and $\bar{m}$ is defined according to the method described by Fig. 3.7.*

Our algorithm does not depend on the way $\mathrm{MS}(\bar{h}, \bar{m})$ is defined. However, this definition provides a correct model for our sequence reconstruction problem.

The score of a match and the padded sequences that support that score, represent the optimum alignment of the participating sites. We are interested in finding a consistent set of matches S with the maximum $\mathrm{Score}(S) = \sum_{\mu \in S} \mathrm{MS}(\mu)$.

REMARK 3.1.

- *Instead of using padded sequences as building blocks of conjecture sequences, we can just as easily use subsequences formed by deleting arbitrary characters from a sequence. The score function for matches and the discussion of consistent match sets remains unchanged under this formulation.*
- *Given a conjecture pair* $(\mathbf{h}, \mathbf{m})$*, if* S *is the set of matches derived from* $(\mathbf{h}, \mathbf{m})$ *then* $Score(S) = Score(\mathbf{h}, \mathbf{m})$.
- *Given a consistent set of matches* S*, we can easily compute a conjecture pair* $(\mathbf{h}, \mathbf{m})$*, such that* $Score(S) = Score(\mathbf{h}, \mathbf{m})$.

According to the last remark, we can formulate an equivalent version of the CSR problem: find a consistent set of matches with maximum total score.

3.2 Simpler versions of the problem

3.2.1 Unambiguous CSR — UCSR

We form a restricted version of the CSR problem by demanding that $\sigma(a, b) = 0$ for $a \neq b$. Moreover, we demand that every letter occurs exactly once in each of $\mathcal{H}$ and $\mathcal{M}$ (counting a^R as an occurrence of a). Suppose we also use subsequences as building blocks for conjectures. Under these restrictions, if we choose a letter for one of the two conjectures, it is determined what letter should match it in the other conjecture, so we call this the *unambiguous* CSR problem, or UCSR. It is easy to see that an optimum solution $(\mathbf{h}, \mathbf{m})$ to an UCSR instance satisfies $\mathbf{h} = \mathbf{m}$, so we can restrict the valid solutions to single sequences from $\mathrm{Conj}(\mathcal{H}) \cap \mathrm{Conj}(\mathcal{M})$ and replace σ with σ' so that $\sigma'(a) = \sigma(a, a)$. This way, for a valid solution $\mathbf{h} = a_1 \dots a_l$ we have $\mathrm{Score}(\mathbf{h}) = \sum_{i=1}^{l} \sigma'(a_i)$.

We can show that UCSR, while simpler, is not easier to approximate than CSR.

LEMMA 3.1. *Let* $\epsilon > 0$*. There exist polynomial time computable functions* τ_0*, and* τ_1 *such that for every instance* X *of CSR*

1. $\tau_0(X)$ *is an instance of UCSR;*
2. *for every solution* $(\mathbf{h}, \mathbf{m})$ *of* X*, there exists a solution* f *of* $\tau_0(X)$ *such that* $Score(f) = Score(\mathbf{h}, \mathbf{m})$*;*
3. *for every solution* x *of* $\tau_0(X)$*,* $\tau_1(x)$ *is a solution of* X *that satisfies* $Score(\tau_1(x)) \geq Score(x)(1-\epsilon)$.

Proof. Let $X = (\mathcal{H}, \mathcal{M}, \sigma)$ be an instance of CSR. Let $\Sigma = \{a_1, \ldots, a_K\}$ be the set of letters that occur in $\mathcal{H} \cup \mathcal{M}$. By replacing multiple occurrences of a letter with distinct replica, and modifying the generator of Score, σ, appropriately, we modify X to an equivalent instance of CSR where each letter occurs in $\mathcal{H} \cup \mathcal{M}$ only once, and no letter occurs in its reversed form. Let $p = \lceil 1/\epsilon \rceil$ and $s = 2pK$. For each letter $a_i \in \Sigma$ and $1 \leq l \leq s$ we define

$$u_l^i = a_{1,l}^i a_{2,l}^i \ldots a_{K,l}^i \text{ and } v_l^i = b_{1,l}^i b_{2,l}^i \ldots b_{K,l}^i.$$

Next, we define

$$w_l^i = \begin{cases} u_l^i v_l^i & \text{if } a_i \text{ occurs in } \mathcal{H} \\ u_l^i (v_{s+1-l}^i)^R & \text{if } a_i \text{ occurs in } \mathcal{M} \end{cases}$$

Finally, $x^i = w_1^i \ldots w_s^i$.

Now we are ready to define $\tau_0(X) = (\mathcal{H}', \mathcal{M}', \sigma')$. We obtain sets $\mathcal{H}'$ and $\mathcal{M}'$ from $\mathcal{H}$ and $\mathcal{M}$ by replacing each a_i with x^i. Next, we reduce the alphabet size of $\tau_0(X)$ by 50% by identifying the following pairs of letters: $a_{j,l}^i$ with $a_{i,l}^j$ and $b_{j,l}^i$ with $b_{i,l}^j$. This identification allows us to define $\sigma'(a_{j,l}^i) = \sigma(a_i, a_j)/s$ and $\sigma'(b_{j,l}^i) = \sigma(a_i, a_j^R)/s$.

To show property 2, consider a solution $(\mathbf{h}, \mathbf{m})$ of X, where $\mathbf{h} = (c_1 \ldots c_L)$ and where $\mathbf{m} = (d_1 \ldots d_L)$. The corresponding solution of $\tau_0(X)$ is the sequence $\rho(c_1, d_1) \ldots \rho(c_L, d_L)$, where $\rho(c, d)$ is the word formed out of all the letters that are common to the replacement words of c and d. More formally, we define

$$\rho(c,d) = \begin{cases} a_{j,1}^i a_{j,2}^i \ldots a_{j,s}^i & \text{if } c = a_i \text{ and } d = a_j, \\ (a_{j,1}^i a_{j,2}^i \ldots a_{j,s}^i)^R & \text{if } c = a_i^R \text{ and } d = a_j^R, \\ b_{j,1}^i b_{j,2}^i \ldots b_{j,s}^i & \text{if } c = a_i \text{ and } d = a_j^R, \\ (b_{j,1}^i b_{j,2}^i \ldots b_{j,s}^i)^R & \text{if } c = a_i^R \text{ and } d = a_j. \end{cases}$$

One can see that this is a valid solution for $\tau_0(X)$. Moreover, we replace a pair of letters (c_i, d_i) with a word of length s, in which every letter scores $\sigma(c_i, d_i)/s$, thus the overall score is unchanged.

Consider a solution f of $\tau_0(X)$. Before we define $\tau_1(f)$ and prove property 3, we will make some preliminary observations. Consider a word x^i that replaced a letter from $\mathcal{H}$ of the original CSR instance X. Because x^i is a sub-word of a word from $\mathcal{H}$, if f contains any letters from x^i , then they form a contiguous sub-word of f, let us call it y^i.

In turn, y^i can be split into subsequences that share letters with various x^js that replaced letter occurrences in $\mathcal{M}$. Again, each such subsequence forms a contiguous sub-word of y^i, say y^i_j, and there are at most K such sub-words. Now we can make an observation about the positions of the letters of y^i within x^i (or of $(x^i)^R$). If two consecutive letters belong to the same y^i_j, their position in x^i differ by at least $2K$, and otherwise they differ by at least 1. However, there can be at most $2K-1$ pairs of the second kind. Therefore if we have $l+(2K-1)+1$ letters in y^i, the positions of the first and the last differ by at least $2Kl+(2K-1)$, hence $2Kl+(2K-1)<2Ks$ and $l \leq s-1$. We can conclude that y^i contains less than $s+2K$ letters.

We define a pair of symbols (letters or reversed letters) (c^i, d^i) such that a symbol a from $\rho(c^i, d^i)$ occurs in y^i and there it has the largest score. Note that $\sigma'(a) > \text{Score}(y^i)/(s+2K)$, thus

$$\sigma(c^i, d^i) > \text{Score}(y^i)\frac{s}{s+2k} = \text{Score}(y^i)(1-\frac{2K}{s+2K}) > \text{Score}(y^i)(1-\epsilon) \tag{3.1}$$

If y^i is empty, then both c^i and d^i are words as well. Finally, we define

$$\tau_1(f = y^{\pi(1)} \dots y^{\pi(K)}) = (c^{\pi(1)} \dots c^{\pi(K)}, d^{\pi(1)} \dots d^{\pi(K)}) = (\mathbf{h}, \mathbf{m})$$

Inequality (3.1) shows that $\text{Score}(\mathbf{h}, \mathbf{m}) > \text{Score}(f)(1-\epsilon)$. ❑

The following theorem is an immediate consequence of Lemma 3.1.

THEOREM 3.1. *If there exists a polynomial time algorithm that solves the UCSR problem with approximation ratio c, then there exists a polynomial time algorithm that solves the CSR problem also with ratio c.*

3.2.2 Consistent Subsets of Integer Pairs — CSoP

We will now show that a very restricted version of UCSR is MAX-SNP hard. In particular, we impose the following restrictions:

1. the alphabet is of the form $\Sigma = \{a_1, \dots, a_{2n}\}$ and $\mathcal{M} = \{a_1 a_2 \dots a_{2n}\}$;

2. $\mathcal{H} = \{a_{i(1)}a_{j(1)}, \dots, a_{i(n)}a_{j(n)}\}$, where the pairs $\{i(k), j(k)\}$ form a partition of $[1, 2n]$ and $i(k) < j(k)$ for $k \in [1, n]$;

3. $\sigma(a_i, a_j) = 1$ if $i = j$ and 0 otherwise.

In those terms the task is to find a set $U \subset \{1, 2, \ldots, 2n\}$ such that if $\{i(k), j(k)\} \subset U$ and $i(k) < l < j(k)$ then $l \notin U$ and such that $|U|$ is maximal. We call this problem Consistent Subsets of Pairs, CSoP. We will show that

THEOREM 3.2. *CSoP is MAX-SNP hard*

Proof. Consider a solution U to an CSoP instance. We say that U is normal if it contains at least one element in each pair $\{i(k), j(k)\}$. Suppose that U is disjoint with an input pair $\{i(k), j(k)\}$ and we try to insert $i(k)$ to U, this insertion can create an invalid solution only if for some k' we have $i(k') \in U$, $j(k') \in U$ and $i(k') < i(k) < j(k')$. In this case we can replace U with $U' = U - \{i(k')\} \cup \{i(k)\}$, $|U'| = |U|$ the number of pairs disjoint with U' is lower. We may conclude that for every solution U there exists a solution U' such that $|U'| = |U|$ and U' intersect every one of the given pairs. We say that U' is a *normal solution*.

To prove our claim, we will reduce 3-MIS to CSoP.

The input to 3-MIS is a 3-regular graph, with $2n$ nodes, a feasible solution is an independent set of nodes, and the goal is to maximize the size of this independent set. Berman and Karpinski [7] have formally shown that 3-MIS is MAX-SNP hard (in folklore, this was an immediate consequence of the original paper on MAX-SNP class by Papadimitriou and Yannakakis [35]). We choose the following representation of the input graph: an $2n \times 3$ matrix A such that $\{i, j\}$ is an edge iff $j \in \{A[i, 1], A[i, 2], A[i, 3]\}$. We also require that the consecutive nodes are never adjacent, i.e., there are no edges of the form $\{i, i + 1\}$ (for $n > 6$ we can order the nodes in such a manner using Dirac's theorem [16]).

In our approximation preserving reduction, the instance translation is as follows: $\mathcal{M} = \{a_1 \ldots a_{10n}\}$; $\mathcal{H} = \mathcal{H}_{\text{nodes}} \cup \mathcal{H}_{\text{edges}}$, where

$$\begin{aligned} \mathcal{H}_{\text{nodes}} &= \{\ \{a_{5i-4}a_{5i}\} :\ 1 \le i \le n\ \} \text{ and} \\ \mathcal{H}_{\text{edges}} &= \{\ \{a_{5i-b}a_{5j-c}\} :\ i < j,\ A[i, b] = j \text{ and } A[j, c] = i\ \}. \end{aligned}$$

Consider a normal solution U. One can show that U contains exactly one element in each *edge pair* $\{5i - b, 5j - c\}$, otherwise there exists node k such that $i < k < j$, hence $5i - b < 5k - 4 < 5k < 5j - c$, hence the *node pair* $\{5k - 4, 5k\}$ is disjoint with U and U is not normal.

Consider now two node pairs that are contained in U, $\{5i-4, 5i\}$ and $\{5j-4, 5j\}$. One can see that no edge connects i and j, otherwise the respective edge pair would be disjoint with U. Define $W = \{i : \{5i-4, 5i\} \subset U\}$. As we observed, W is an independent set. Moreover, the size of U equals $5n + |W|$.

Consider now an idependent set W. For every edge e there exists an endpoint of e, say $i(e)$ such that $i(e) \in W$, we can assume that $e = \{i(e), A[i(e), b(e)]\}$. We can form a normal solution with $5n + W$ elements as follows

$$\{5i : \ i \text{ is a node } \} \cup \{5i(e) - b(e) : \ e \text{ is an edge } \} \cup \{5i - 4 : \ i \in W\}.$$

Therefore this is an approximation preserving reducibility. ❑

3.2.3 Reducing CSR to 1-CSR

We now consider 1-CSR, i.e., the CSR problem with the following restriction: set $\mathcal{M}$ consists of exactly one sequence. We will show the following theorem.

THEOREM 3.3. *If there exists an approximation algorithm $\mathcal{A}$ that solves 1-CSR with approximation ratio r, then there exists an approximation algorithm $\mathcal{A}'$ that solves CSR with approximation ratio $2r$.*

Proof. For $\mathcal{F} = \{u_1, \ldots, u_n\}$ we define $\mathcal{F}' = \{u_1 \ldots u_n\}$, a set containing only the concatenation of all words from $\mathcal{F}$, in some arbitrary order. The algorithm $\mathcal{A}'$ processes the input instance of CSR, $(\mathcal{H}, \mathcal{M}, \sigma)$, as follows: it runs $\mathcal{A}$ twice, on $(\mathcal{H}, \mathcal{M}', \sigma)$ and on $(\mathcal{M}, \mathcal{H}', \sigma)$, and selects the better of the two solutions.

Let $\mathrm{Opt}(X)$ be the score of the optimum solution for instance X. It will suffice to show that

$$\mathrm{Opt}(\mathcal{H}, \mathcal{M}', \sigma) + \mathrm{Opt}(\mathcal{M}, \mathcal{H}', \sigma) \geq \mathrm{Opt}(\mathcal{H}, \mathcal{M}, \sigma) \tag{3.2}$$

because inequality (3.2) implies that the solution found by $\mathcal{A}'$ has a score that is at least

$$\max(\frac{1}{r}\mathrm{Opt}(\mathcal{H}, \mathcal{M}', \sigma), \frac{1}{r}\mathrm{Opt}(\mathcal{M}, \mathcal{H}', \sigma)) \geq \frac{1}{2r}\mathrm{Opt}(\mathcal{H}, \mathcal{M}, \sigma)$$

Assume that $\mathcal{H} = \{h_1, \ldots, h_k\}$ and $\mathcal{M} = \{m_1, \ldots, m_{k'}\}$. For convenience, assume that each letter appears only once in $\mathcal{H} \cup \mathcal{M}$. Consider an optimum solution for $(\mathcal{H}, \mathcal{M}, \sigma)$, i.e. a pair of words $\mathbf{h} = s_{\pi(1)} \ldots s_{\pi(k)} \in \mathrm{Conj}(\mathcal{H})$ and $\mathbf{m} = t_{\pi(1)} \ldots t_{\pi(k')} \in \mathrm{Conj}(\mathcal{M})$, where $s_{\pi(i)}$ is a (possible

reversed) padded sequence of h_i (and similarly for t_i). Given that $(\mathbf{h}, \mathbf{m}) = (a_1 \dots a_l, b_1 \dots b_l)$ for some l and $\text{Score}(\mathbf{h}, \mathbf{m}) = \sum_{i=1}^{l} \sigma(a_i, b_i)$, we will also view this solution as the sequence of pairs $(a_1, b_1), \dots, (a_l, b_l)$. We will paint these pairs with two colors, say blue and yellow, and then we assemble a solution of $(\mathcal{H}, \mathcal{M}', \sigma)$ from all the blue pairs, and a solution of $(\mathcal{M}, \mathcal{H}', \sigma)$ from all the yellow pairs. By the very construction, the scores of these two solutions add to $\text{Score}(\mathbf{h}, \mathbf{m})$.

The coloring of pairs requires some preliminary steps. First, we tag every letter in s_j with j, and we do the same with t_j. This way, a pair (a_i, b_i) has a pair of tags, say (j, j'). If a pair has tag (j, j'), we will say that j is an $\mathcal{H}$-partner of j', and j' is an $\mathcal{M}$-partner of j. The partners can be ordered as follows: if for some $i < i'$ the pairs (a_i, b_i) and $(a_{i'}, b_{i'})$ have tags (j, j_1) and (j, j_2), then j_1 is an earlier $\mathcal{M}$-partner of j than j_2; similarly, if these tags are (j_1, j) and (j_2, j), then j_1 is an earlier $\mathcal{H}$-partner of j than j_2. We will paint the tags, and each will have the color of its tag. Our coloring rules are the following:

- If j' is the first $\mathcal{M}$-partner of j, then we paint the tag (j, j') with blue.
- If j is the first $\mathcal{H}$-partner of j', then we paint the tag (j, j') with yellow.

A blue solution is formed in a simple manner: we rearrange the blue pairs (and reverse them if necessary) in such a way that their b's form a subsequence of $m_1 \dots m_{k'}$. To show that we have formed a correct solution for $(\mathcal{H}, \mathcal{M}', \sigma)$ it remains to show that the pairs with a's from a particular s_j form a contiguous part of this solution, and that they are ordered as in h_j. If such a pair is present, then, because it is blue, it has a tag (j, j') where j' is the first $\mathcal{M}$-partner of j. The pairs with tag (j, j') form a contiguous part of pairs with b's from $t_{j'}$. When we form the blue solution from the original one, we cut the original solution into pieces corresponding to various t's, reverse if necessary, remove the yellow pairs and reassemble in a new order. During this process, the fragment with tags (j, j') remains contiguous and keeps its original ordering.

The yellow solution for $(\mathcal{M}, \mathcal{H}', \sigma)$ is formed identically. To show (3.2) it suffices to show that each tag is painted, so each $\sigma(a_i, b_i)$ will be added to the score of one of the two solutions (if a tag is painted with two colors, then $\sigma(a_i, b_i)$ is added to both scores, thus increasing the left hand side of (3.2)). Consider a tag (j, j'). The positions of a_i's from s_j form an integer interval,

say $[d, e]$, and positions of b_i's from $t_{j'}$ form another interval, say $[d', e']$. Thus a pair (a_i, b_i) has tag (j, j') if and only if $\max(d, d') \leq i \leq \min(e, e')$. One can see that if $d' \leq d$ then j' is the first $\mathcal{M}$-partner of j', and if $d \leq d'$, then j is the first $\mathcal{M}$-partner of j. In the former case tag (j, j') is blue, and in the latter it is yellow. ❑

3.2.4 1-CSR and Interval Selection Problem — ISP

A 1-CSR problem instance has the form $(\mathcal{H}, m, \sigma)$. Because each fragment of $\mathcal{H}$ is involved in at most one match, we can assume that in each match the site from $\mathcal{H}$ is full. Thus each match in a solution can be described as $(k, [i, j])$, which denotes pair $(h_k, m(i, j))$. Selecting such a match yields profit $\mathrm{MS}(h_k, m(i, j))$.

We can reduce 1-CSR to a more abstract Interval Selection Problem, ISP for short, where we are given set A of integer intervals and a non-negative profit function $p : [1, k] \times A \to R^+$. The task is to select at most one interval of A for each $i \in [1, k]$, so that the selected intervals are disjoint and the sum of profits is maximal. ISP was studied in the context of scheduling [2] by Bar-Noy *et al.* [3], who described an algorithm with ratio 2. Later Berman and DasGupta [6] described a *Two Phase Algorithm* that obtains ratio 2 and runs in time $O(n \log n)$, where $n = k|A|$.

Our reduction defines as A the set of all subintervals of $[1, |m|]$ and for each fragment $h_i \in \mathcal{H}$ sets $p(i, [d, e]) = \mathrm{MS}(h_i, m(d, e))$. Clearly an approximation algorithm for ISP yields an algorithm for 1-CSR with exactly the same approximation ratio.

Corollary 3.1. *There exists a polynomial time algorithm for the CSR problem with approximation factor 4.*

3.3 Approximation algorithms for CSR

3.3.1 Iterative improvements

We will maintain the solution to a CSR problem instance as a consistent set of matches. To form tools for solving the general problem, we will first describe how to search for one type of matches only i.e. only border matches or only full matches. The algorithms we use there are

[2] More general versions of ISP are considered with different A_i for each $i \in [1, k]$

selected in such a way that later we will be able to combine them into an algorithm that searches for both types of matches. We tackle different versions of the problem in the following manner:

- We define an iterative *improvement algorithm*
  - The algorithm is defined by set $\mathcal{I}$ of *improvement methods*, i.e. a finite set of routines that have a constant number of parameters of the form $f(i, j)$ where f is a fragment. For $\mathcal{R} \in \mathcal{I}$, and a parameter vector p, an *improvement attempt* $I = \mathcal{R}(p)$ changes the current solution by discarding some matches and making some new matches.
  - *gain*(I), the gain of an improvement attempt I, is the increase in total score after the improvement attempt I; if a given I is not applicable to the current legal set then $\text{gain}(I) = 0$.
  - The algorithm starts with an empty set of matches and makes improvement attempts with positive gain until none exists.

- If the scores are large numbers, gains can be comparitively very small and we cannot easily bound the running time. To ensure that our algorithm runs in polynomial time, we use the scaling method described by Chandra and Halldórsson [11] as follows. We first use the simple algorithm of Corollary 3.1 to obtain a solution with score X. We run the local improvement algorithm after truncating the weights of all match scores to integer multiples of X/k^2, where k is an upper bound on the number of possible matches. Since the score of the optimal solution is at most $4X$ and each local improvement has gain at least X/k^2, the number of improvements is limited to $4k^2$. This approach underestimates the optimal solution by at most X/k and hence, increases the approximation ratio by a factor $(k+1)/k$. Thus, when we prove an approximation ratio τ, the ratio actually proven has the form $\tau(k+1)/k$ or $\tau + \epsilon$. However, in practice, alignment scores should have few precision bits and this step should not be necessary.

- In the analysis, we use the optimum solution, *Opt*, and the set of matches generated by our algorithm, L, to define a collection of improvement attempts, $\mathcal{J}$. $\mathcal{J}$ is constructed such that each attempt $I \in \mathcal{J}$ removes matches $\phi(I) \subseteq L$ and creates matches $\omega(I) \subseteq \mathit{Opt}$. Since the

algorithm has terminated, no improvement has a positive gain which leads to the inequality

$$\sum_{I \in \mathcal{J}} \text{Score}(\omega(I)) \leq \sum_{I \in \mathcal{J}} \text{Score}(\phi(I))$$

By showing that the score of each match of *Opt* appears in the term on the left exactly n_1 times and that the score of each match of L appears on the right at most n_2 times, we have $n_1 \times \text{Score}(Opt) \leq n_2 \times \text{Score}(L)$. In this manner we prove that the algorithm defined by the set of improvement methods has approximation ratio $\frac{n_2}{n_1} + \epsilon$.

In defining the improvement methods and in the subsequent analysis we use the following notions:

- The *solution graph* of a set of matches S is the bipartite graph $(\mathcal{H} \cup \mathcal{M}, \mathcal{E})$, where $\{h, m\} \in \mathcal{E}$ iff S contains a match with sites in h and m.
- The connected components of this graph are called *islands*.
- In an island that consists of one fragment only, the fragment is *simple*.
- In an island that consists of two fragments only, one of the fragments is *simple* and the other *multiple*.
- In other islands, fragments that participate in a single match are *simple* and fragments that participate in more than one match are *multiple*.

DEFINITION 3.5. *In the following definitions f is a fragment, $\bar{f}, \breve{f}$ are sites in f, S a consistent set of matches and F a set of fragments*

- *Mult(S) is the set of all multiple fragments of S.*
- *Simp(S) is the set of all simple fragments of S.*
- *Site $f(i,j)$ is contained in $f(i',j')$ if $i' \leq i \leq j \leq j'$.*
- *Site $f(i,j)$ is adjacent to $f(i',j')$ if $i' = j+1$ or $j' = i-1$.*
- *For $f \in$ Mult(S), Match($\bar{f}, S$) $= \{g | (g, \breve{f}) \in S$ and $\breve{f}$ is contained in $\bar{f}\}$. We can extend the definition to sets of sites.*
- *Cb(f, S), the contribution of the fragment f to the solution S, is the sum of scores of all match scores in S involving f. The contribution of a set of fragments F is the sum Cb(F, S) $= \sum_{f \in F}$ Cb(f, S).*

- $\widehat{S}_{\mathcal{H}}$ *is the set if all sites of fragments of* $\mathcal{H}$ *that participate in matches of* S. $\widehat{S}_{\mathcal{M}}$ *is defined similarly.* $\widehat{S} = \widehat{S}_{\mathcal{H}} \cup \widehat{S}_{\mathcal{M}}$.
- $f(i,j)$ *is* hidden *by* $f(i',j')$ *if* $i' < i \leq j < j'$, *if* $f(i',j') \in \widehat{S}$, *then we also say that* $f(i,j)$ *is hidden by* S. *Note that if* $(f, \bar{g}) \in S$, *then only the border sites of* f *are not hidden.*

3.3.2 Full CSR

In Full CSR problem we are limiting the legal solutions to a given CSR instance to those that contain full matches only.

Consider the solution graph of a solution to a Full CSR problem instance. Because each match in this solution contains a full site, for each edge in our graph one of the ends has one neighbor only. Consequently, in each island, at most one node is a multiple fragment.

Our improvement methods create new full matches using Two Phase Algorithm, TPA(B, S), where

- B is a union of sites of $\mathcal{M}$ and defines
  Sites$(B) = \{\bar{m} :\ \bar{m}$, viewed as an interval, is contained in $B\ \}$;
- S is the current solution and is used to define the profit function
  $p(h, \bar{m}) = \mathrm{MS}(h, \bar{m}) - \mathrm{Cb}(h, S)$.

We run TPA(B, S) with index set $\mathcal{H}$, interval set Sites(B) and profit function p. In our algorithms TPA(B) is a shorthand for TPA(B, S) where S is the current solution.

LEMMA 3.2. *Let Opt be any optimal solution. A run of TPA*(B, S) *creates a set of matches with the sum of scores at least*

$$hope(B, S) = \frac{1}{2} \sum_{f \in \mathrm{Match}(B, Opt)} (Cb(f, Opt) - Cb(f, S)).$$

Proof. From the definition of the profit function, we see that each fragment $f \in \mathrm{Match}(B, Opt)$ participates with score $\mathrm{Cb}(f, Opt) - \mathrm{Cb}(f, S)$. Since TPA has approximation ratio 2 the result follows. ❑

LEMMA 3.3. *If for each fragment of* $\mathcal{H} \cup \mathcal{M}$ *we know whether it is simple or multiple in Opt, then Full CSR has a 2-approximation algorithm.*

Proof. Let $\mathcal{H} = \mathcal{H}_{\text{simp}} \cup \mathcal{H}_{\text{mult}}$, where $\mathcal{H}_{\text{simp}} = \mathcal{H} \cap \text{Simp}(Opt)$. Similarly let $\mathcal{M} = \mathcal{M}_{\text{simp}} \cup \mathcal{M}_{\text{mult}}$. Let S be the set of matches generated by the following algorithm

1. Run TPA($\mathcal{H}_{\text{mult}}$) with index set $\mathcal{M}_{\text{simp}}$.
2. Run TPA($\mathcal{M}_{\text{mult}}$) with index set $\mathcal{H}_{\text{simp}}$.

Since no fragment is involved in both TPA runs, from Lemma 3.2 it follows that

$$\begin{aligned}
\text{Score}(S) &\geq \frac{1}{2}\text{Cb}(\text{Match}(\mathcal{H}_{\text{mult}}), Opt) + \frac{1}{2}\text{Cb}(\text{Match}(\mathcal{M}_{\text{mult}}), Opt) \\
&\geq \frac{1}{2}(\text{Cb}(\mathcal{M}_{\text{simp}}, Opt) + \text{Cb}(\mathcal{H}_{\text{simp}}, Opt)) \\
&\geq \frac{1}{2}\text{Cb}(\text{Simp}(Opt), Opt) \\
&\geq \frac{1}{2}\text{Score}(Opt)
\end{aligned}$$

❑

The algorithm of Lemma 3.3 can be used only if the role that each fragment plays in *Opt* is known. Since this information is not generally available, the rest of this section describes a more complicated iterative improvement algorithm.

Let S be the current set of matches. In the improvement methods described below, a site $\bar{f}$ may need to be *prepared* for a match. The manner of *preparation* of the site depends on the classification of f:

- $f \in \text{Simp}(S)$: detach f from its match (if any).
- $f \in \text{Mult}(S)$: if the site is hidden by S it cannot be prepared (and the improvement that specifies a match of $\bar{f}$ cannot proceed). Otherwise, restrict any match of the form $(g, \breve{f})$ to $(g, \breve{f} - \bar{f})$. Note that if $\breve{f}$ is contained in $\bar{f}$, g becomes completely detached.

Our iterative algorithm *Full Improve* has one improvement method.

$I_1(f, \bar{g}, \breve{g})$

If $\breve{g}$ contains $\bar{g}$ and is not hidden by S,

1. Prepare the sites f and $\breve{g}$.
2. Match f with $\bar{g}$ (intuitively we plug in f to site $\bar{g}$).

3. Run TPA($\breve{g} - \bar{g}$).
4. If g is detached from some site $\bar{f}_1$ during preparation of $\breve{g}$ in Step 1, run TPA($\bar{f}_1$).

We say that $\bar{g}$ is the target of this improvement attempt.

Fig. 3.8 shows two cases of an I_1 improvement attempt. Only Steps 1-3 are executed in the first case. Step 4 is executed in the second case because g is detached during preparation.

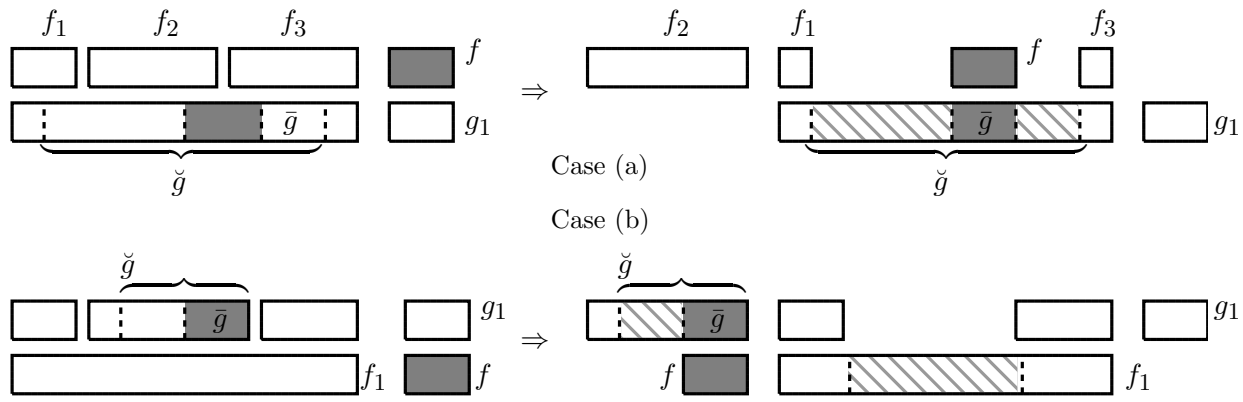

Fig. 3.8. Two cases (a),(b) of improvement attempt $I_1(f, \bar{g}, \breve{g})$ are shown. In both cases preparation of f detaches it from g_1. In (a) preparation of $\breve{g}$ detaches f_2 and restricts matches made by f_1, f_3. In (b) preparation of $\breve{g}$ detaches g from f_1. Sites on which TPA is run are filled with slanted lines.

Let L be the consistent set of matches generated by our improvement algorithm. In the analysis that follows, we assume that the optimum legal solution is *Opt*.

Definition 3.6. *Suppose* $g \in$ *Mult*(*Opt*).

- *If* $(\bar{f}, \bar{g}) \in L$ *and* $\bar{g}$ *hides* $\breve{g} \in \widehat{Opt}$ *we say that* f owns TPA site $\breve{g}$. *This defines "owns" and "TPA sites".*
- *A* TPA zone *is the union of adjacent TPA sites.*

- *If* $(\bar{f}, \bar{g}) \in Opt$ *and* $\bar{g}$ *is not part of any TPA zone, then* $\bar{g}$ *is a* plug-in site. *See Fig. 3.9.*

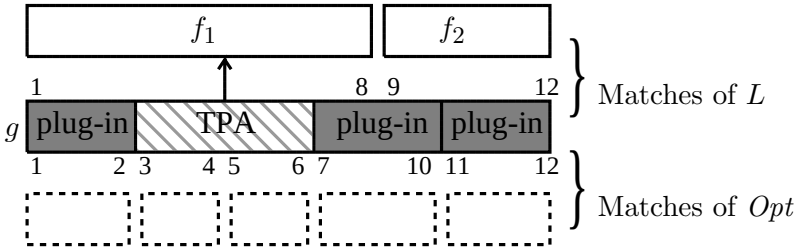

Fig. 3.9. $g \in \text{Mult}(L)$ is partitioned into TPA zones and plug-in sites. Arrows point to the owners. Matches of Opt are shown using fragments with dashed outlines. $g(1,8) \in \widehat{L}$ hides $g(3,4), g(5,6) \in \widehat{Opt}$. Therefore $g(3,6) = g(3,4) \cup g(5,6)$ is a TPA zone owned by f_1. $g(1,2), g(7,10), g(11,12)$ are not hidden by any site of $\widehat{L}$ so they are plug-in sites.

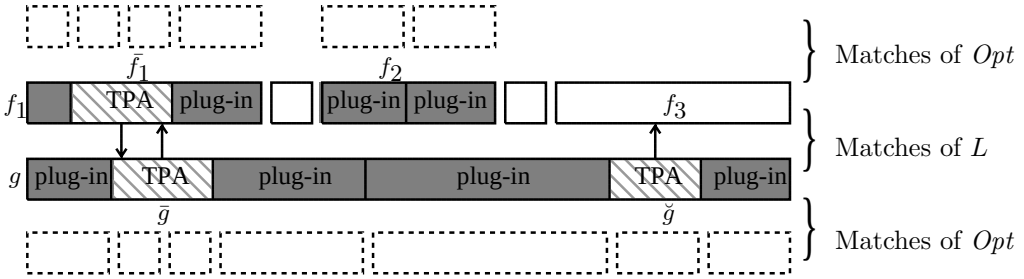

Fig. 3.10. TPA zones associated with plug-in sites. Fragments $f_1, f_2, g \in \text{Mult}(Opt)$ are partitioned into TPA zones and plug-in sites. TPA zone $\bar{f}_1$ is associated with the two plug-in sites adjacent to it because $f_1 \in \text{Simp}(L)$. TPA zone $\bar{g}$ of $g \in \text{Mult}(L)$ is not associated with its neighbors because its owner $f_1 \in \text{Mult}(Opt)$. TPA zone $\breve{g}$ of $g \in \text{Mult}(L)$ is associated with its neighbors because its owner $f_3 \in \text{Simp}(Opt)$.

REMARK 3.2. *The following remarks can be easily verified.*

- *Each TPA zone has a single owner.*
- *A TPA zone always extends between two plug-in sites (but not vice versa).*
- *Any fragment of Mult(Opt) is partitioned into TPA zones and plug-in sites.*
- *Any fragment of Simp(L) ∩ Mult(Opt) has at most one TPA zone.*

DEFINITION 3.7. *TPA zone* $\bar{g}$ is associated with *plug-in site* $\breve{g}$ *if*

- $\bar{g}$ *is adjacent to* $\breve{g}$ *and*
- $g \in Simp(L)$ *or*
  $g \in Mult(L)$ *but some* $f \in Simp(Opt)$ *owns* $\bar{g}$. *See Fig. 3.10.*
- *zone*($\breve{g}$) *is the union of plug-in site* $\breve{g}$ *and the TPA zones associated with it.*

We now construct the collection (multi-set) of improvement attempts $\mathcal{J}$. Each attempt of $\mathcal{J}$ targets a plug-in site. For a plug-in site $\bar{g}$, we will use the short hand notation $I_1(\bar{g})$ to denote the attempt $I_1(f, \bar{g}, \text{zone}(\bar{g}))$, where $(f, \bar{g}) \in Opt$. For each plug-in site $\bar{g}$, $\mathcal{J}$ has one improvement attempt $I_1(\bar{g})$.

In our analysis, we now estimate gain($\mathcal{J}$). In the improvement attempts of the algorithm a match is either made explicitly in Step 2 of an I_1 improvement method or as a result of a TPA run. The construction of $\mathcal{J}$ ensures that the explicit matches attempted are matches of *Opt* and that TPA runs are made only on TPA zones of multiple fragments. Therefore, using hope as a lower bound on the sum of scores of matches created by a TPA run, we can compute a lower bound of gain($\mathcal{J}$) in which the positive terms are scores of matches of *Opt* and the negative terms are scores of matches of L. We show that in this estimate of gain($\mathcal{J}$)

- the score of each match of *Opt* is added exactly once
- the score of each match of L is subtracted at most 3 times

Since no improvement attempt has a positive gain we then have

$$0 \geq \text{gain}(\mathcal{J}) \geq \text{Score}(Opt) - 3 \times \text{Score}(L) \tag{3.3}$$

and the required approximation ratio follows.

LEMMA 3.4. *TPA is applied to each TPA zone exactly twice.*

Proof. Let $\bar{g}$ be a TPA zone.

- If $\bar{g}$ is associated with the neighboring two plug-in sites, the I_1 attempts that target these plug-in sites run TPA on $\bar{g}$ in Step 3.
- Otherwise, $\bar{g}$ is owned by some $f \in \text{Mult}(Opt)$ and f has two plug-in sites. The improvement attempts that target these plug-in sites, detach f from g and run TPA on $\bar{g}$ in Step 4.

❑

LEMMA 3.5. *The score of each match of Opt is added exactly once in gain*$(\mathcal{J})$.

Proof. Consider any match $(f, \bar{g}) \in Opt$. Since $g \in \text{Mult}(Opt)$ can be partitioned into plug-in sites and TPA zones there are two possiblilities:

- $\bar{g}$ is a plug-in site: the attempt $I_1(\bar{g}) \in \mathcal{J}$ so $\text{MS}(f, \bar{g})$ is added once to the estimate of $\text{gain}(\mathcal{J})$.
- $\bar{g}$ is a TPA site: according to Lemma 3.4 two TPA runs are made on the TPA zone $\breve{g}$ containing $\bar{g}$. Since $f \in \text{Match}(\breve{g}, Opt)$ it follows from Lemma 3.2 that together these runs add $\text{Cb}(f, Opt) = \text{MS}(f, \bar{g})$ to the estimate of $\text{gain}(\mathcal{J})$.

❑

Note that since

$$\text{Score}(L) = \sum_{f \in \text{Simp}(L)} \text{Cb}(f, L) = \sum_{f \in \text{Mult}(L)} \text{Cb}(f, L) \tag{3.4}$$

all the negative terms in the estimate of $\text{gain}(\mathcal{J})$ can be expressed in the form $-(hl_f + apl_f + ppl_f)\text{Cb}(f, L)$ where

- hl_f, or *hope loss of* f, is caused by the use of $\text{Cb}(f, L)$ in the estimates of hope;
- apl_f, or *active preparation loss of* f, is caused by detaching f during preparation of sites on fragment f itself;
- ppl_f, or *passive preparation loss of* f, is caused by detaching f during preparation of sites on other fragments.

LEMMA 3.6. *For $f \in Simp(Opt)$ we have $hl_f + apl_f = 1$.*

Proof. Suppose $f \in \text{Simp}(Opt)$, i.e., $(f, \bar{g}) \in Opt$ for some $\bar{g}$. There are two cases to consider:

- $\bar{g}$ is a plug-in site: $apl_f = 1$ because f is detached in Step 1 of the attempt $I_1(\bar{g}) \in \mathcal{J}$.
- $\bar{g}$ is a TPA site: By Lemma 3.4 two TPA runs are made on the TPA zone containing this TPA site; $hl_f = 1$ because (by Lemma 3.2) f contributes $-\frac{1}{2}\text{Cb}(f, L)$ to the hope of each TPA run, thus $hl_f = \frac{1}{2} + \frac{1}{2}$.

❑

LEMMA 3.7. *For $f \in Mult(Opt) \cap Simp(L)$ we have $hl_f + apl_f = 2$.*

Proof. Clearly, $hl_f = 0$. From Remark 3.2 it follows that f has exactly two border plug-in sites, say $\bar{f}, \breve{f}$. The improvements attempts $I_1(\bar{f})$, $I_1(\breve{f})$ detach f during preparation. So $apl_f = 2$. ❑

LEMMA 3.8. *The score of each match of L is lost at most 3 times in gain($\mathcal{J}$).*

Proof. By Equation 3.4 it suffices to show that for every $f \in \text{Simp}(L)$, $hl_f + apl_f + ppl_f \leq 3$.

Consider the match $(f, \bar{g}) \in L$ and suppose $g \in \text{Simp}(Opt)$. By Lemma 3.6, $hl_g + apl_g = 1$. Equivalently this loss of $\text{Cb}(g, L)$ can be construed as a single passive preparation loss for each $f \in \text{Match}(g, L)$ because $\text{Cb}(g, L) = \sum_{f \in \text{Match}(g,L)} \text{Cb}(f, L)$. Therefore, $ppl_f = 1$ and using Lemma 3.6, Lemma 3.7 we have $hl_f + apl_f + ppl_f \leq 3$.

Now consider the match $(f, \bar{g}) \in L$ when $g \in \text{Mult}(Opt)$. We need to consider the 4 cases that are illustrated in Fig. 3.11.

Suppose that $f \in \text{Mult}(Opt)$, by Lemma 3.7 it suffices to show that $ppl_f \leq 1$. There are two cases:

- if f owns a TPA zone $\breve{g}$ the attempts that target the two plug-in sites adjacent to $\breve{g}$ can restrict the match. But as Fig. 3.11 shows for f_1, the restricted portions are non-overlapping and together subtract the score of the match at most once. So $ppl_f \leq 1$.
- if $\bar{g}$, the match of f (f_2 in Fig. 3.11) is contained within a plug-in site $\breve{g}$ then $ppl_f = 1$ because of the improvement attempt $I_1(\breve{g})$.

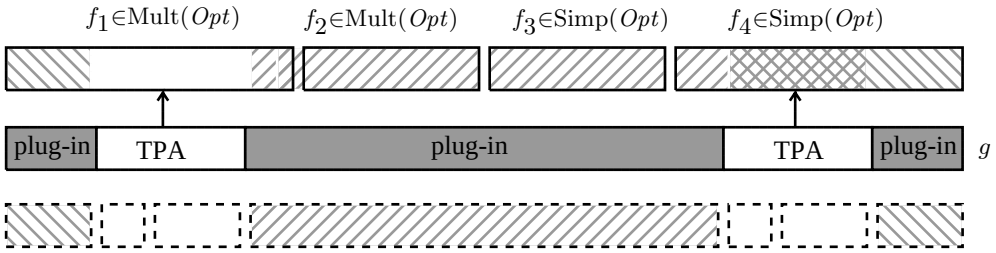

Fig. 3.11. The four types of matches in which $g \in \text{Mult}(Opt) \cap \text{Mult}(L)$ participates in L. Matches are restricted(removed) by I_1 attempts. Shading is used to denote the portion of the match restricted (removed) and the fragment that plugs in and is responsible for the restriction (removal).

Suppose that $f \in \text{Simp}(Opt)$, by Lemma 3.6 it suffices to show that $ppl_f \leq 2$. There are two cases:

- if $\bar{g}$, the match of f (f_3 in Fig. 3.11) is contained within a plug-in site $\breve{g}$ then $ppl_f = 1$ because of the improvement attempt $I_1(\breve{g})$.
- if f (f_4 in Fig. 3.11) owns a TPA zone $\breve{g}$, $\breve{g}$ is associated with the two plug-in sites adjacent to it. Attempts that target these plug-in sites can restrict the match. These restrictions may overlap but together amount to at most twice the score of $(f, \bar{g})$. Thus, $ppl_f \leq 2$.

❑

THEOREM 3.4. *Algorithm Full Improve solves the Full CSR problem with approximation factor* $3 + \epsilon$.

Proof. The approximation ratio follows from inequality 3.3, Lemma 3.5 and Lemma 3.8. ❑

3.3.3 Border CSR

In Border CSR problem we consider problem instances where the optimum solution contains border matches only.

LEMMA 3.9. *There exists a polynomial time algorithm for the Border CSR problem with approximation factor 2.*

Proof. Consider the solution graph $(\mathcal{H} \cup \mathcal{M}, \mathcal{E})$ of the optimum solution to a Border CSR instance $(\mathcal{H}, \mathcal{M}, \sigma)$. Because each fragments has only two borders and every site must be a border site, this is a degree 2 bipartite graph. Thus, we can partition $\mathcal{E}$ into two matchings $\mathcal{A}$ and $\mathcal{B}$. Clearly the sum of the scores of the matches represented by one of the sets $\mathcal{A}$ or $\mathcal{B}$ is at least 50% of the total score. In this set each fragment participates in at most one match; thus we can assume that all sites in the matches are full.

Consequently, to find a legal solution to our instance of Border CSR with score at least 50% of the optimum, it suffices to find an optimum legal solution in which every match consists of two full sites. The latter we can find by applying the algorithm for the maximum weight matching to the bipartite graph with node set $\mathcal{H} \cup \mathcal{M}$ and edge weight function

$$w(\{h, m\}) \;=\; \mathrm{MS}(h, m).$$

❑

However, we prefer an alternate algorithm with approximation ratio 3, *Border Improve*, for the Border CSR problem. Unlike the algorithm of Lemma 3.9, we will be able to combine this algorithm with the algorithm for the Full CSR problem discussed in the previous section.

The algorithm for the Full CSR problem creates full matches only. Therefore, each island of the solution contains at most one multiple fragment. We call such islands *1-islands*. The algorithm for the Border CSR problem allows each multiple fragment to participate in at most one border match. So, in addition to 1-islands, the solution may contain *2-islands* — islands with two multiple fragments sharing a border match.

Since all sites chosen by the Border Improve algorithm are border sites, we will occasionally refer to them simply as sites in this section. The algorithm repeatedly prepares chosen sites on pairs of fragments and forms border matches. A site is prepared as described in the previous section. In addition, if the site belongs to the multiple fragment of a 2-island, we first *break* the 2-island by removing the match between the two multiple fragments. This ensures that our solution consists of 1-islands and 2-islands only. We have two improvement methods.

$I_2(\bar{f}, \bar{g})$. Prepare the sites $\bar{f}, \bar{g}$ and match them.

$I_3(\bar{f}_1, \bar{g_1}, \bar{f}_2, \bar{g_2})$. Applicable if f_1, g_1 are multiple fragments of the same 2-island. Prepare all four sites. Make the matches $(\bar{f}_1, \bar{g_2})$ and $(\bar{g_1}, \bar{f}_2)$.

Fig. 3.12 shows a sample I_3 improvement attempt.

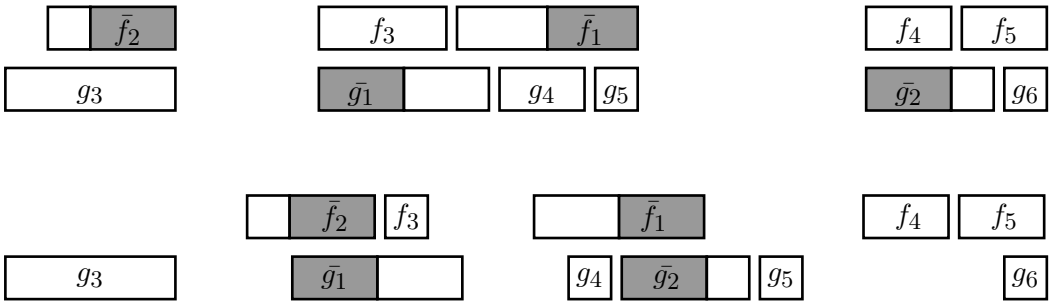

Fig. 3.12. $I_3(\bar{f}_1, \bar{g_1}, \bar{f}_2, \bar{g_2})$ improvement attempt breaks the 2-island formed by f_1, g_1 and the 2-island formed by f_5, g_2. The matches in which f_3, g_4 participate are restricted while the matches in which f_2, f_4, g_5 participate are removed during preparation.

Let L be the solution generated by Border Improve. With the knowledge of the the optimum solution, *Opt*, we will construct a multi-set of improvement attempts $\mathcal{J}$. Each improvement attempt in $\mathcal{J}$ removes some matches of L and creates matches of *Opt*. $\mathcal{J}$ will have the property that if all the improvement attempts in it are carried out

- each match of L will be removed 12 times
- each match of *Opt* will be attempted 4 times

When the algorithm terminates because all attempts fail, adding the inequalities representing the failure of attempts of $\mathcal{J}$ gives us

$$12 \times \text{Score}(L) \geq 4 \times \text{Score}(Opt) \tag{3.5}$$

which is the required result.

All the improvement attempts in $\mathcal{J}$ try to form matches present in *Opt*. Thus, in the description of $\mathcal{J}$ an attempt of method I_2 is described as $I_2(\bar{f})$, the other site being implicit. Similarly, an attempt of method I_3 is specified as $I_3(\bar{f}, \bar{g})$, where f, g are the multiple fragments of the same 2-island.

We call the sites that are specified in an attempt the *explicit parameters* and the sites that are implied the *implicit parameters*. We construct $\mathcal{J}$ as follows:

- Let f be any simple fragment or multiple fragment in a 1-island of L. Let f^1, f^2 be the border sites of f in *Opt*. Then $\mathcal{J}$ contains two improvement attempts $I_2(f^1)$, two improvement attempts $I_2(f^2)$.
- Let f, g be the two multiple fragments of a 2-island in L. Let f^1, f^2 be the border sites of f and let g^1, g^2 be the border sites of g. Then $\mathcal{J}$ contains the four improvement attempts – $I_3(f^1, g^1)$, $I_3(f^1, g^2)$, $I_3(f^2, g^1)$ and $I_3(f^2, g^2)$.

LEMMA 3.10. *The score of each (border) match of Opt is added exactly 4 times in gain*($\mathcal{J}$).

Proof. From the construction of $\mathcal{J}$ described above, it is easy to see that each border site of *Opt* is the explicit parameter of an I_2 or I_3 improvement attempt exactly twice. Because this applies to both sites of a border match, each match of *Opt* is attempted 4 times. ❑

LEMMA 3.11. *The score each border match of L is lost at most 12 times in gain*($\mathcal{J}$).

Proof. Consider the border match formed by the two multiple fragments, f, g, in a 2-island. Let f^1, f^2 be the border sites of h and let g^1, g^2 be the border sites of g in *Opt*.

- The match between f and g is broken 4 times because of the improvement attempts $I_3(f^1, g^1)$, $I_3(f^1, g^2)$, $I_3(f^2, g^1)$, $I_3(f^2, g^2)$. These are the only attempts of $\mathcal{J}$ in which f^1, f^2, g^1, g^2 are explicit parameters.
- Each of the sites f^1, f^2, g^1, g^2 is an implicit parameter of two improvement attempts in $\mathcal{J}$. These 8 improvement attempts break the 2-island during the preparation of the concerned site.

Thus, the score of the match is lost 12 times overall. ❑

LEMMA 3.12. *The score each full match of L is lost at most 12 times in gain($\mathcal{J}$).*

Proof. Consider any full match $(f, \bar{g}) \in L$. Since $hl_f = 0$ it suffices to show that $apl_f + ppl_f \leq 12$. Let f^1, f^2 be the border sites of f and let g^1, g^2 be the border sites of g in *Opt*.

- The sites f^1, f^2 participate in 4 attempts each. $apl_f = 8$ because these attempts detach f from g during preparation.
- As indicated by Fig. 3.13 the 4 attempts in which g^1 participates restrict the portion of the match represented by site f^3. Similarly, the 4 attempts in which g^2 participates restrict the the portion of the match represented by site f^4. $ppl_f = 4$ because overall these restrictions subtract the score of the match exactly 4 times.

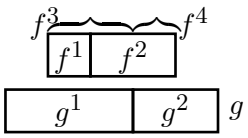

Fig. 3.13. $(f, \bar{g}) \in L$ is shown with the border sites that the fragments have in *Opt*. f^3, f^4 represent the portion of the match restricted by attempts involving g^1, g^2 respectively.

Thus, the match loses its score at most 12 times overall. ❑

THEOREM 3.5. *Algorithm Border Improve solves the Border CSR problem with approximation factor $3 + \epsilon$.*

Proof. The proof follows from inequality 3.5, Lemma 3.10, Lemma 3.11 and Lemma 3.12. ❑

3.3.4 General CSR

We now consider the general CSR problem. A site is prepared in exactly the same manner as in Section 3.3.3. Thus, the solution generated by the algorithm consists of 1-islands and 2-islands only.

The iterative improvement algorithm, *CSR Improve*, consists of method I_1 from Section 3.3.2 and methods I_2, I_3 from Section 3.3.3. I_2, I_3 are modified by treating the border sites as targets of I_1 attempts. Thus, for each border site an additional site that contains the border site needs to be specified. Since the modifications are similar for both methods, only I_2 is explained in detail.

$I_2(f^1, f^2, g^1, g^2)$

Applicable if f^1, g^1 are border sites contained in f^2, g^2 respectively

1. Prepare f^2, g^2.
2. Match the border sites f^1, g^1.
3. If f was detached from some site $\bar{g_1}$ in Step 1,
   run TPA($\{\bar{g_1}, g^2 - g^1\}$) else run TPA($g^2 - g^1$).
4. If g was detached from some site $\bar{f_1}$ in Step 1,
   run TPA($\{\bar{f_1}, f^2 - f^1\}$) else run TPA($f^2 - f^1$).

Fig. 3.14 shows a sample I_2 improvement attempt.

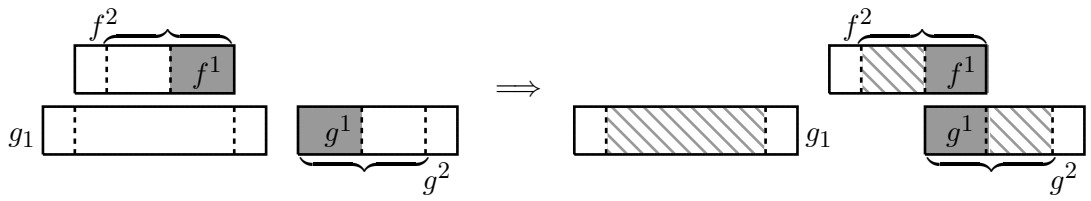

Fig. 3.14. In $I_2(f^1, f^2, g^1, g^2)$ improvement attempt, f is detached from g_1 when the site f^2 is prepared. After the shaded border sites are matched, TPA is run on the sites filled with slanted lines.

Also, if an I_2 or I_3 attempt breaks a 2-island during preparation, the attempt can be combined with an I_1 attempt that targets the newly exposed border site (or part of it). Fig. 3.15 shows a valid combination of attempts.

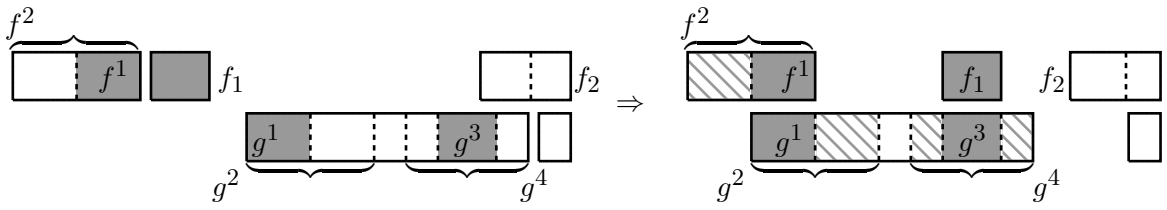

Fig. 3.15. $I_2(f^1, f^2, g^1, g^2)$ attempt breaks the 2-island formed by g and f_2. $I_1(f_1, g^3, g^4)$ can be combined with it. TPA runs are made on the sites filled with slanted lines.

Let L be the solution generated by the CSR Improve algorithm and let *Opt* be some optimal solution. We can partition every fragment $f \in \text{Mult}(Opt)$ into plug-in sites and TPA zones using Definition 3.6.

We can now construct a collection of improvement attempts $\mathcal{J}$. The I_2, I_3 attempts in $\mathcal{J}$ try to form border matches of *Opt*. An I_2 attempt of $\mathcal{J}$ with one explicit parameter, say $I_2(\bar{f})$, represents the attempt $I_2(\bar{f}, \text{zone}(\bar{f}), \bar{g}, \text{zone}(\bar{g}))$ where $(\bar{f}, \bar{g}) \in Opt$ is a border match.

Similarly, I_3 attempts can be described by 2 explicit parameters – the border plug-in sites of the two multiple fragments of the same 2-island. $\mathcal{J}$ is constructed as follows:

1. Suppose $f, g \in \text{Mult}(Opt)$ are the multiple fragments of a 2-island in L. Let f^1, f^2 be the border sites of f and let g^1, g^2 be the border sites of g. Then $\mathcal{J}$ contains the four improvement attempts – $I_3(f^1, g^1)$, $I_3(f^1, g^2)$, $I_3(f^2, g^1)$ and $I_3(f^2, g^2)$.
2. Suppose $\bar{f}$ is a plug-in site that participates in a border match in *Opt* and is not used as a parameter in Step 1, then $\mathcal{J}$ has two improvement attempts $I_2(\bar{f})$.

3. Suppose $\bar{f}$ is a plug-in site that participates in a full match in *Opt*, then $\mathcal{J}$ has four improvement attempts $I_1(\bar{f})$. Whenever possible, these I_1 improvement attempts are combined with the I_2, I_3 attempts of Steps 1,2.

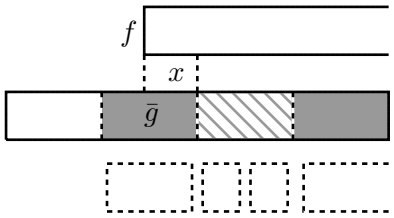

Fig. 3.16. f, g are the multiple fragments of a 2-island. Plug-in sites are shaded and TPA zones are filled with slanted lines. Only some of the matches are shown for clarity.

LEMMA 3.13. *The score of each match of Opt is added exactly 4 times in gain*$(\mathcal{J})$.

Proof. Consider a match $(\bar{f}, \bar{g}) \in Opt$ where $g \in \text{Mult}(Opt)$. g can be partitioned into plug-in sites and TPA zones. There are now 3 possibilities:

- $(\bar{f}, \bar{g})$ is a border match: Lemma 3.10 shows that $\text{MS}(\bar{f}, \bar{g})$ is added 4 times in $\mathcal{J}$.
- $\bar{g}$ is an inner plug-in site and $\bar{f} = f$: the 4 $I_1(\bar{g})$ attempts added in Step 3 of the construction of $\mathcal{J}$ ensure that $\text{MS}(f, \bar{g})$ is added 4 times in $\mathcal{J}$.
- $\bar{g}$ is a TPA site and $\bar{f} = f$: Lemma 3.4 and the above two items imply that TPA is run on the TPA zone containing $\bar{g}$ exactly 8 times. From Lemma 3.2 it is easy to see that $\text{Cb}(f, Opt) = \text{MS}(f, \bar{g})$ is added 4 times.

❑

LEMMA 3.14. *The score of each match of L is lost at most 12 times in gain*$(\mathcal{J})$.

Proof. It follows from Lemma 3.11 that the score of each border match of L is lost at most 12 times.

One special instance that needs to be handled is shown in Fig. 3.16. As mentioned, the match of the 2-island formed by the multiple fragments f, g is broken 12 times because of I_2, I_3 attempts. However, the portion of the match marked x is also restricted in the four $I_1(\bar{g})$ attempts of $\mathcal{J}$. To prevent x from being lost 16 times overall it is necessary to combine $I_1(\bar{g})$ attempts with I_2, I_3 attempts that break the 2-island.

Analysis along the lines of Lemma 3.8 can now be used to show that each full match of L loses its score at most 12 times. ❑

THEOREM 3.6. *Algorithm CSR Improve solves the CSR problem with approximation ratio* $3 + \epsilon$.

Proof. The proof follows from Lemma 3.13 and Lemma 3.14. ❑

Chapter 4

Saturn: a heuristic for 1-CSR

In this chapter, we assume that one of the genomes is assembled i.e., we have a set of m-contigs and a single h-contig, h. We consider the problem of ordering and orienting the m-contigs based on their alignments with h. Our formulation, described in Section 4.1, is well suited to handle real data sets generated by local alignment tools in a natural manner. We describe an exact branch-and-bound algorithm in Section 4.2 and our heuristic *Saturn* in Section 4.3.

To test our heuristic, we used BLASTZ [43] to compute the alignments of mouse contigs of *arachne.3* assembly [4] with the assembled human chromosome 22 sequence. The mouse contigs were then ordered and oriented using our heuristic. In this test, our heuristic computed a solution that is within 5% of the optimum. We also show that the solution score correlates with improved biological information.

4.1 Problem Statement

We assume that each local alignment identifies a region of a m-contig, m_i (or its reverse complement m_i^R), a region of h, and the alignment score. We model these alignments as *hits*. More formally,

DEFINITION 4.1. *A* hit*, p, aligns interval $[b_m, e_m]$ of some contig m_l^d (where d is blank or R) with interval $[b_h, e_h]$ of contig h with score s. We use the notation*

- $owner(p) = m_l$
- $dir(p) = d$
- $m\text{-}interval(p) = [b_m(p), e_m(p)] = [b_m, e_m]$
- $h\text{-}interval(p) = [b_h(p), e_h(p)] = [b_h, e_h]$
- $\sigma(p) = s$

We view the input to our problem as a set of hits D. Our solution, a subset of D, will represent a collection of "consistent alignments". The definitions that follow formalize this notion.

DEFINITION 4.2. *We say that hit p* precedes *hit q ($p \prec q$) if*

- $owner(p) = owner(q)$
- $dir(p) = dir(q)$
- $e_m(p) < b_m(q)$ *and* $e_h(p) < b_h(q)$

Also, $p \preceq q$ if $p \prec q$ or $p = q$.

DEFINITION 4.3. *The* hit precedence graph *is the weighted directed acyclic graph $G = (D, E, w)$ where*

- $\forall p, q \in D, (p, q) \in E$ *iff* $p \prec q$
- $\forall p \in D, w(p) = \sigma(p)$

DEFINITION 4.4. *In the hit precedence graph $G = (D, E, w)$*

- *a path,* P*, is a sequence of vertices $\langle p_1, \ldots, p_n \rangle$ s.t. $\forall i < n, (p_i, p_{i+1}) \in E$.*
- *the* weight *of P, $w(P)$, is the sum of the weights of the vertices in P*
- *$maxpath(p, q)$ is the maximum weight path that starts at p and ends at q*

Using dynamic programming we can easily compute $maxpath(p, q)$, for every pair of hits p, q which satisfy $p \prec q$.

DEFINITION 4.5. *If $p \preceq q$, we define* match $\mathbf{p} = match(p, q)$ *with the following attributes:*

- $owner(\mathbf{p}) = owner(p)$
- $dir(\mathbf{p}) = dir(p)$
- $h\text{-}interval(\mathbf{p}) = [b_h(p), e_h(q)]$
- $m\text{-}interval(\mathbf{p}) = [b_m(p), e_m(q)]$
- $\sigma(\mathbf{p}) = w(maxpath(p, q))$

For a set of matches A, we can extend the above notation in a straight forward manner - using addition for numerical values, union otherwise. In particular,

- $owner(A) = \{owner(\mathbf{p}) \mid \mathbf{p} \in A\}$
- $h\text{-}interval(A) = \{h\text{-}interval(\mathbf{p}) \mid \mathbf{p} \in A\}$
- $\sigma(A) = \sum_{\mathbf{p} \in A} \sigma(\mathbf{p})$

DEFINITION 4.6. *Two matches* $\mathbf{p}$, $\mathbf{q}$ *are* compatible *if*

- $owner(\mathbf{p}) \neq owner(\mathbf{q})$
- $h\text{-}interval(\mathbf{p}) \cap h\text{-}interval(\mathbf{q}) = \phi$

A set of matches is compatible if the matches are pairwise compatible.

Our problem is the following: given D, find a compatible set of matches with maximum score. It follows from Definition 4.5 that we can compute in polynomial time the set of all matches defined by D. So we can reduce our problem to a simpler one : given a set of matches $\mathcal{M}$ find the compatible subset with maximum score. Following the notation used in the previous chapter, we call this the 1-CSR problem. The 1-CSR problem can be solved with approximation ratio 2 by the *Two Phase Algorithm* [6] in time $O(n \log n)$, where $n = \mid h\text{-}interval(\mathcal{M}) \mid$.

Every pair of matches in a valid solution satisfies two properties - different owners and disjoint *h-interval*s. If we relax the first property, the problem reduces to that of computing the weighted maximum independent set in the interval graph $G = (V, E, w)$, where

- $V = h\text{-}interval(\mathcal{M})$
- $E = \{(v_1, v_2) \in V \times V \mid v_1 \cap v_2 \neq \phi\}$
- $\forall \mathbf{p} \in \mathcal{M}, w(h\text{-}interval(\mathbf{p})) = \sigma(\mathbf{p})$

The weighted maximum independent set problem in intervals graphs can be solved in $O(n \log n)$ time [23]. We let $MIS(\mathcal{M})$ denote the optimal solution to this relaxed version of the 1-CSR problem.

REMARK 4.1. *Let $\mathcal{M}$ be a 1-CSR instance, $TPA(\mathcal{M})$ the solution returned by the Two Phase Algorithm (TPA) and $Opt(\mathcal{M})$ the optimal solution.*

1. $\sigma(TPA(\mathcal{M})) \leq \sigma(Opt(\mathcal{M})) \leq \sigma(MIS(\mathcal{M}))$
2. *If each contig participates in exactly one match then* $\sigma(MIS(\mathcal{M})) = \sigma(Opt(\mathcal{M}))$

It follows from the previous remark that $lb(\mathcal{M})$, $ub(\mathcal{M})$ defined as $\sigma(TPA(\mathcal{M}))$, $\sigma(MIS(\mathcal{M}))$ respectively are valid lower and upper bounds on $\sigma(Opt(\mathcal{M}))$.

4.2 Branch and Bound algorithm

We now describe an exact algorithm that will later be used to find optimal solutions to sub-problems as part of a more involved heuristic.

We can solve a 1-CSR problem instance by examining all compatible sets of matches and selecting the set with highest score. However, explicitly enumerating all elements of the search space of feasible solutions could take exponential time. We use the classic branch and bound method to eliminate from consideration parts of the search space where no optimal solution can exist.

To apply the branch-and-bound method we

- divide the search space of the problem to create smaller sub-problems
- compute upper and lower bounds for each sub-problem and if possible eliminate it from further consideration

The method starts by considering the original problem with the complete search space – this is the root problem. The search space is then divided into two parts. The problems aimed at finding the optimal solution for each part become the children of the root search node. Applying the division procedure recursively, we generate a tree of subproblems.

Given a 1-CSR problem instance, $\mathcal{M}$, we can represent a node, u, of the tree as a pair (S, C), where

- $S, C \subseteq \mathcal{M}$, $S \cap C = \phi$
- S is a compatible set of matches

- each match of C is individually compatible with S i.e. $\forall a \in C, S \cup \{a\}$ is a feasible solution

Let $\mathcal{U}$ represent the set of all feasible solutions. The node u represents the subset R_u of solutions that satisfy

$$R_u = \{S' \in \mathcal{U} \mid S \subseteq S' \subseteq S \cup C\}$$

For a node $u = (S, C)$ our division procedure create subproblems v, w as follows:

1. let a be the match in C with maximum score
2. let incompat$(a) = \{b \in \mathcal{M} \mid a$ is incompatible with $b\}$
3. $v = (S \cup \{a\}, C - \text{incompat}(a)), w = (S, C - \{a\})$

Note that $R_v = \{S' \in R_u \mid a \in S'\}$ while $R_w = \{S' \in R_u \mid a \notin S'\}$. Since $R_u = R_v \cup R_w$ this division procedure ensures that we do not overlook any feasible solutions.

To generate all feasible solutions, we can start with a root node $(\phi, \mathcal{M})$ and apply the recursive procedure until all leaf nodes are of the form (S, ϕ). The key to avoid generating some subtrees whose leaves contain only sub-optimal solutions is the following observation: if (S', ϕ) is a leaf in a subtree rooted at (S, C) then $\sigma(S') \leq \sigma(S) + ub(C)$. So, if $\sigma(S) + ub(C) \leq \sigma(B)$ where B is some known feasible solution, we can prune the entire subtree rooted at (S, C).

The actual implementation uses a queue data structure to carry out a breadth first traversal of the search tree and is described in Figure 4.1.

4.3 Algorithm *Saturn*

In this section we describe criteria to

1. eliminate matches that cannot be in any optimal solution
2. select matches that are guaranteed to be in some optimal solution

We use these techniques till saturation (no more progress can be made) and solve the remaining problem by using *TPA* or by greedily selecting compatible matches. In our tests, we found that the majority of the solution is obtained using the above criteria. We, therefore, obtain a better result than that obtained by using *TPA* alone.

```
function Exact(M)
    B = TPA(M) is a feasible solution
    Initialize queue Q with the root node (φ, M)
    while (Q is not empty)
        let u = (S, C) be the first element of Q
        if σ(S) + lb(C) > σ(B)
            B = S ∪ TPA(C)
        if σ(S) + ub(C) > σ(B)
            compute v, w the children of u
            add v, w to the queue Q
    end
    return B
end
```

Fig. 4.1. Pseudo-code for the branch-and-bound algorithm. To prevent combinatorial explosion, in our implementation we ensure that the function returns a *null* solution if the size of the queue Q exceeds 10,000.

4.3.1 Removing dominated matches

DEFINITION 4.7. *A contig is considered* single *if it is the owner of exactly one match of* $\mathcal{M}$ [1]. *Let* $singles(\mathcal{M})$ *denote the matches of* $\mathcal{M}$ *that are owned by singles.*

Because we keep eliminating matches, $\mathcal{M}$ will stand for the set of matches that are not yet eliminated. Definitions based on $\mathcal{M}$ are thus dynamic and need to be recomputed when $\mathcal{M}$ changes.

DEFINITION 4.8. *A match* $\mathbf{p}$ *is* dominated *by a compatible set of matches* A *if*

- $\forall \mathbf{q} \in A$, $owner(\mathbf{q})$ *is either a single or the same as* $owner(\mathbf{p})$
- $\forall \mathbf{q} \in A$, $h\text{-}interval(\mathbf{q}) \subseteq h\text{-}interval(\mathbf{p})$

[1] *As we remove matches from consideration, an increasing number of contigs will become single.*

- $\sigma(\mathbf{p}) < \sigma(A)$

A match is dominated if there exists a compatible set of matches that dominates it.

LEMMA 4.1. *The optimal solution does not contain any dominated matches.*

Proof. Let S be any optimal solution. Suppose $\mathbf{p} \in S$ is dominated by a compatible set of matches A. Then it is easy to verify that $S - \{\mathbf{p}\} \cup A$ is a compatible set of matches with score greater than $\sigma(S)$. ❑

DEFINITION 4.9. *Let A be any set of matches. Then A* restricted to (b, e) *is defined as* $A(b, e) = \{\mathbf{p} \in A \mid h\text{-}interval(\mathbf{p}) \subseteq [b, e]\}$.

Let $S = Opt(singles(\mathcal{M})) = MIS(singles(\mathcal{M}))$. Then it is easy to see that a match $\mathbf{p} \in \mathcal{M}$ is dominated if

- $\sigma(\mathbf{p}) < \sigma(S(b_h(\mathbf{p}), e_h(\mathbf{p})))$ or
- $\exists \mathbf{q} \in \mathcal{M}$ that satisfies
  1. $owner(\mathbf{q}) = owner(\mathbf{p})$,
  2. $h\text{-}interval(\mathbf{q}) \subseteq h\text{-}interval(\mathbf{p})$, and
  3. $\sigma(\mathbf{p}) < \sigma(S(b_h(\mathbf{p}), b_h(\mathbf{q}) - 1)) + \sigma(\mathbf{q}) + \sigma(S(e_h(\mathbf{q}) + 1, e_h(\mathbf{p})))$

Since the number of matches can be quadratic in the number of hits, say n, the method of removing dominated matches suggested by the above statement has $O(n^4)$ time complexity. In practice, few contigs have more than 10 hits and the time taken is almost linear in n. Also, since this method is used only once, its running time does not have a significant impact on the overall performance of the heuristic.

Later, whenever the set of singles changes, we use the simpler check for dominated matches described in Figure 4.2.

```
procedure FilterDominatedMatches(M)
    let S = Opt(singles(M))
    for each match p ∈ M
        if σ(p) ≤ σ(S(b_h(p), e_h(p))) then
        M = M − {p}
    end
end
```

Fig. 4.2. Removing matches dominated by singles.

4.3.2 Checking for local solutions

Here we formulate a sufficient condition for recognizing that a solution to a sub-problem is contained in an optimal solution for the full problem.

DEFINITION 4.10. *The* overlap components *of $\mathcal{M}$ are the connected components of the graph $G = (\mathcal{M}, E)$ where*

$$(\mathbf{p}, \mathbf{q}) \in E \text{ iff } h\text{-interval}(\mathbf{p}) \cap h\text{-interval}(\mathbf{q}) \neq \phi$$

An overlap-closed set *is the union of one or more overlap components.*

DEFINITION 4.11. *Contig m_k is* local *to a set of matches A if*

$$\forall \mathbf{p} \in \mathcal{M}, owner(\mathbf{p}) = m_k \Longrightarrow \mathbf{p} \in A$$

Also, we say that A has a local solution *if all matches in $Opt(A)$ have owners that are local to A.*

LEMMA 4.2. *If an overlap-closed set C has a local solution then*

$$Opt(\mathcal{M}) = Opt(C) \cup Opt(\mathcal{M} - C)$$

Proof. Clearly $\sigma(Opt(\mathcal{M})) \leq \sigma(Opt(C)) + \sigma(Opt(\mathcal{M} - C))$. To prove equality it suffices to show that $Opt(C) \cup Opt(\mathcal{M} - C)$ is a feasible solution. Let $\mathbf{p} \in Opt(C), \mathbf{q} \in Opt(\mathcal{M} - C)$ be any two matches. Then,

- $owner(\mathbf{p}) \neq owner(\mathbf{q})$ because $owner(\mathbf{p})$ is local to C
- $h\text{-}interval(\mathbf{p}) \cap h\text{-}interval(\mathbf{q}) = \phi$ because the overlap component containing $\mathbf{p}$ is a subset of C

By Definition 4.6, $\mathbf{p}$ is compatible with $\mathbf{q}$. Since $\mathbf{p}$, $\mathbf{q}$ were chosen arbitrarily from $Opt(C)$, $Opt(\mathcal{M} - C)$ it follows that $Opt(C) \cup Opt(\mathcal{M} - C)$ is a compatible set of matches. ❑

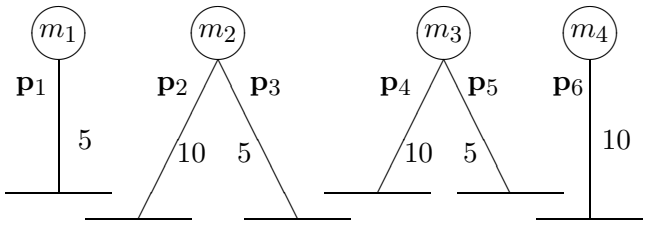

Fig. 4.3. A 1-CSR problem instance with 3 overlap components $C_1 = \{\mathbf{p}_1, \mathbf{p}_2\}$, $C_2 = \{\mathbf{p}_3, \mathbf{p}_4\}$ and $C_3 = \{\mathbf{p}_5, \mathbf{p}_6\}$

Lemma 4.2 suggests that we can solve a 1-CSR instance by repeatedly identifying overlap-closed sets with local solutions and solving them using the Exact algorithm described earlier in Section 4.2. For example, consider the 1-CSR problem instance $\mathcal{M} = \{\mathbf{p}_1, \ldots, \mathbf{p}_6\}$ shown in Figure 4.3. The optimal solutions to the overlap components are $Opt(C_1) = \{\mathbf{p}_2\}$, $Opt(C_2) = \{\mathbf{p}_4\}$ and $Opt(C_3) = \{\mathbf{p}_6\}$. C_3 is the only overlap components with a local solution. But once this sub-problem is solved and the matches of C_3 are removed, $\mathcal{M}$ reduces to $C_1 \cup C_2$. Now, C_2 has a local solution and $Opt(C_2)$ can be added to the optimal solution. Finally, we are left only with C_1 and then $Opt(C_1)$ can also be added to the optimal solution. Function LocalSolutions shown in Figure 4.4 generalizes this approach.

We first call the function LocalSolutions with the set of overlap components. When the procedure terminates, no overlap component has a local solution. At this point, we can try to

```
function LocalSolutions(𝒞)
    𝒞 is a collection of disjoint overlap-closed sets
    Compute the directed graph G where
        V(G) = 𝒞
        for each C ∈ V(G)
            if Opt(C) cannot be computed by algorithm Exact
                (C, C) ∈ E(G)
            else
                ∀C' ≠ C, (C, C') ∈ E(G) iff owner(Opt(C)) ∩ owner(C') ≠ φ
        end
    while (some C ∈ 𝒞 has out-degree 0)
        S ← S ∪ Opt(C)
        remove C with all incident edges from G
    end
    return (S, V(G))
end
```

Fig. 4.4. Procedure for identifying overlap-closed sets with local solutions and solving them separately. S is the optimal solution computed and $V(G)$ consists of overlap-closed sets whose matches form the residual problem.

form larger overlap-closed sets and check if these larger sets have local solutions. In our implementation, we simply collapse connected components of the graph into large overlap-closed sets, and use the LocalSolution procedure again. However, this method does not guarantee that we will make any progress. For instance, consider the 1-CSR instance of Figure 4.5. None of the overlap components have a local solution, and the graph G at the end of the procedure has 4 edges – $(C_1, C_2), (C_2, C_1), (C_3, C_4), (C_4, C_3)$. But once we collapse the connected components, we still do not have any vertices with out-degree 0: we now have a graph with edges $(C_1 \cup C_2, C_3 \cup C_4), (C_3 \cup C_4, C_1 \cup C_2)$.

We interrupt the merging process when the optimal solutions to the overlap-closed sets can no longer be computed by the branch-and-bound algorithm. At this point, we can choose compatible matches greedily or use *TPA* to find an approximate solution to the residual problem.

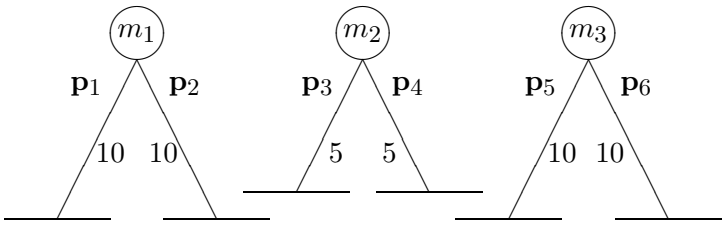

Fig. 4.5. A 1-CSR instance with overlap components $C_1 = \{\mathbf{p}_1\}$, $C_2 = \{\mathbf{p}_2, \mathbf{p}_3\}$, $C_3 = \{\mathbf{p}_4, \mathbf{p}_5\}$ and $C_4 = \{\mathbf{p}_6\}$.

4.4 Results

Our first data set was created by using BLASTZ [43] to compare approximately 700,000 mouse (*Mus musculus*) contigs of the *arachne.3* assembly [4] with the human chromosome 22 sequence/ The resulting dataset consisted of 45,115 mouse contigs participating in 69,858 hits with the human

genomic sequence. The distribution of hits among contigs is shown in Table 4.1. The distribution of hit scores is shown in Table 4.2.

Number of Hits	Number of Contigs
1	33,224
2	6,640
3	2,568
4	1,083
5	546
6	359
7	211
8	149
9	115
10	69
11	43
12	32
13	24
14	20
15	4
16	7
17	5
18	4
19	5
20–30	6
31–40	1

Table 4.1. Distribution of hits among contigs.

We now describe the performance of our algorithm *Saturn* on this data set.

1. Generating matches from hits : Since each contig owning n hits (with the same direction) can result in $n(n+1)/2$ matches, this step could have generated 112,554 matches. However, in

Score range	Number of Hits
3000–4000	36,702
4000–5000	7,768
5000–6000	4,579
6000–7000	2,942
7000–8000	2,083
8000–9000	1,617
9000–10000	1,262
10000–20000	7,093
20000–30000	2,668
30000–40000	1,239
40000–50000	720
50000–60000	458
60000–70000	301
70000–80000	140
80000–90000	86
90000–100000	60
100000–200000	119
200000–300000	17
300000–400000	3
400000–500000	1

Table 4.2. Distribution of hit scores

our implementation we integrate the check for dominated matches with the match generation procedure. As a result we only have 74,722 matches at the end of this step and Step 2 is somewhat accelerated.

2. Removing dominated matches : Whenever the set of singles changes, we recompute $Opt(singles(\mathcal{M}))$ and check for dominated matches using the FilterDominatedMatches procedure described in Section 4.3.1. The total number of matches eliminated from consideration using this procedure is only 2,420. However, eliminating matches allows some contigs to become local to an

overlap component and creates sub-problems that can be solved independently thus increasing the effect of Step 3.

3. Checking for local solutions : The set of matches at the end of Step 1 forms 14,000 overlap components. By using the LocalSolutions procedure along with our merging method for forming larger overlap-closed sets, we solve a substantial part of the problem. The score of the solution computed at the end of this step is 185,523,674. This is the unambiguous part of the overall solution to the problem instance.

4. Finishing using *TPA*: The residual problem consists of 10,657 contigs owning 23,646 matches that form 1,416 overlap components. The upper bound on the value of the optimal solution to this sub-problem (computed using MIS) is 49,575,953. We use *TPA* to find a solution with score 38,970,987. Thus, an upper bound on the value of the overall optimal solution is ub = 185,523,674 + 49,575,953 = 235,099,627, while our solution has score 185,523,674 + 38,970,987 = 224,494,661. In other words our solution score is within 5% of the optimum.

We also computed solutions using two other methods:

- a simple greedy algorithm, *Greedy*, that orders the contigs by placing each contig at a position corresponding to the *h-interval* of its maximum scoring hit
- applying *TPA* to the set of matches at the end of Step 1

The performance of all three algorithms is summarized in Table 4.3.

It is clear from Table 4.3 that for the given problem instance, algorithm *Saturn* succeeds in obtaining a solution with score close to the optimum. However, we also need to verify that this success corresponds to improved biological information. Since our method is based on the hypothesis that conserved regions tend to occur in the same order in related species, we decided to check if mouse contigs that were adjacent in a solution were part of the same mouse chromosome or not.

The three solutions together contain 21,501 distinct contigs. We used MEGABLAST [49] to align these mouse contigs with the masked assembled mouse sequences released by Ensembl on May

	Saturn	*TPA*	*Greedy*
size	16193	17507	19748
score	95.49	90.90	84.22
chr prediction	16.8	14.8	13.3

Table 4.3. Performance of the three algorithms. The first row shows the number of contigs that are ordered and oriented by each solution. The score of each algorithm as a percentage of the upper bound *ub* is shown in the second row. The last row shows the percentage of contigs that are adjacent in the solution and map to the same mouse chromosome.

2, 2002. Only 80% of the contigs had strong enough hits that enabled us to place them precisely. 10% of the contigs had high scoring hits with multiple chromosomes or had strong hits with different directions to the same chromosome. The remaining 10% of the contigs had no strong hits at all![2] The last row of Table 4.3 is based on our mapping of contigs to chromosomes. While it is clear that the biological information does correlate with the score of a solution, the overall prediction is not good. However, the prediction quality improves dramatically when low scoring matches are removed from the solution (see Table 4.4).

Min score	*Saturn*	*TPA*	*Greedy*
4000	49.3	46.3	42.2
5000	65.9	63.4	59.7
6000	75.9	74.3	71.3
7000	81.2	79.8	77.8

Table 4.4. Effect of low scoring matches on prediction quality. Solutions are computed by removing matches with score less than the number shown in the first column.

[2]This suggests that the mouse assembly is incomplete or that the mouse contigs of *arachne.3* assembly are contaminated by human sequence.

We also created an "ideal" set of contigs by fragmenting the assembled mouse genome sequence into 10 kbp pieces. We used MEGABLAST to compute high scoring alignments of these contigs with the human chromosome 22 sequence. The resulting data set consisted of 1,627 contigs participating in 3,373 hits. We then ordered the contigs using the three different algorithms described earlier. Contigs that were adjacent in the solutions produced by *Saturn/Greedy*, were within 50 kbp of each other in the assembled mouse genome approximately 60% of the time. This number improved to 75% when 200 lowest scoring contigs that together accounted for only 3% of the overall solution score were removed from the solution. It is clear that the minimum cut-off score can affect the quality of the solution. The choice of this cut-off score depends on the alignment algorithm as well as the pairs of species.

Chapter 5

Conclusions

The number of species with publicly available sequence data is expected to grow rapidly in the next few years. However, it is clear that only a few model species will be fully sequenced. In most cases, sequence data will remain in the form of numerous contigs. In this thesis, we examined the problem of ordering and orienting contigs based on alignments with contigs of a related species. In particular, we formulated the order/orient problem as a maximization problem, showed that it is MAX-SNP hard and described a factor 3 approximation algorithm. We also designed a heuristic that computes high scoring solutions for real data sets.

The performance of our heuristic is influenced by several parameters - the degree of sequence conservation between the two species, the quality and size of the contigs, the presence of duplicated regions and gene families, the choice of an alignment tool etc. The following are some suggestions for future work:

- We tuned the *Saturn* heuristic for a particular data set of the 1-CSR problem. For problem instances that involve species separated by large distances, the biological premise of the heuristic does not hold. On the other hand, for closely related species such as human and chimp, even algorithm *Greedy* will find solutions with high score. In such cases, we can still use *Saturn* to estimate the upper bound and, thus, gauge the quality of the solution produced by *Greedy*. One should expect a solution with score within 98-99% of the optimum for these data sets. Experiments with data sets obtained from different pairs of species are necessary to identify a wider range of techniques.

- The *arachne.3* data set consists of ∼700,000 contigs. We examined the alignments of these contigs with 8 assembled human chromosomes and discovered that ∼250,000 contigs had alignments with more than one chromosome. This is to be expected because gene families

consisting of extremely similar genes are spread across multiple chromosomes and contigs with alignments to one member of the family are likely to also align with the other members of the family. If we were to solve multiple 1-CSR problem instances (one for each chromosome) independently, a contig could appear in more than one 1-CSR solution. We can also consider all chromosomes simultaneously, but this would create very large overlap closed sets with no local solutions. To resolve such troublesome situations, it may be necessary to first partition the set of all contigs using maximum matching.

- We consider all alignments generated by *blastz* in our heuristic. It is, however, possible that not all alignments are equally important. In [27] two conserved regions are designated *syntenic anchors* if their alignment is the only significant match that either region has with the other species. Several runs of more than 120 anchors with the same order and orientation were observed in mouse chromosome 16 and homologous segments of the human chromosomes. The 1-CSR problem remains MAX-SNP hard even if the set of regions participating in alignments is restricted to the set of syntenic anchors. However, it is possible that solutions computed using such data sets have more biological significance.

The techniques described in this thesis provide a foundation for future work in the above directions.

References

[1] M. D. Adams et al. The genome sequence of *Drosophila melanogaster*. *Science*, 297:2185–2195, 2000.

[2] S. Anderson. Shotgun dna sequencing using cloned dnase i-generated fragments. *Nucleic Acids Research*, 10:3015–3027, 1981.

[3] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating the throughput of multiple machines in real-time scheduling. *Proc. 31st ACM STOC*, pages 622–631, 1999.

[4] S. Batzoglou et al. Arachne: a whole-genome shotgun assembler. *Genome Research*, 12(1):177–189, January 2002.

[5] D. Benson, I. Karsch-Mizrachi, et al. Genbank. *Nucleic Acids Research*, 30(1):17–20, 2002.

[6] P. Berman and B. DasGupta. Multi-phase algorithms for throughput maximization for real-time scheduling. *J. Combinatorial Optimization*, 4(3):307–323, 2000.

[7] P. Berman and M. Karpinski. On some tighter inapproximability results. *ICALP* '99. Full version in ECCC Report 29, University of Trier, 1998.

[8] J. Bouck, W. Miller, J. Gorrell, D. Muzny, and R. A. Gibbs. Analysis of the quality and utility of random shotgun sequencing at low redundancies. *Genome Research*, 8:1074–1084, 1998.

[9] W. Castle and W. Wachter. Variation of linkage in rats and mice. *Genetics*, 9:1–12, 1924.

[10] The Sanger Center & The Genome Sequencing Center. Towards a complete human genome sequence. *Genome Research*, 8:1097–1108, 1998.

[11] B. Chandra and M. M. Halldórsson. Greedy local improvement and weighted set packing approximation. *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 169–176, 1999.

[12] International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, February 2001.

[13] The *C. elegans* Sequencing Consortium. Genome sequence of the nematode c. elegans: a platform for investigating biology. *Science*, 282(5396):2012–2018, December 1998.

[14] A. Coulson, J. Sulston, S. Brenner, and J. Karn. Toward a physical map of the genome of the nematode *Caenorhabditis elegans*. *Proc. Nat. Acad. Sci. USA*, 83:7821–7825, 1986.

[15] P. L. Deininger. Random subcloning of sonicated dna: application to shotgun dna sequence analysis. *Anal. Biochem*, 129:216–223, 1983.

[16] G. A. Dirac. Some theorems on abstract graphs. *Proc. London Math. Soc.*, 2:69–81, 1952.

[17] J. Ehrlich, D. Sankoff, and J. H. Nadeau. Synteny conservation and chromosome rearrangements during mammalian evolution. *Genetics*, 147(1):289–296, September 1997.

[18] R. C. Gardner et al. The complete nucleotide sequence of an infectious clone of cauliflower mosaic virus by m13mp7 shotgun sequencing. *Nucleic Acids Research*, 9:2871–2888, 1981.

[19] A. Goffeau et al. The yeast genome directory. *Nature*, 387:S1–S105, May 1997.

[20] E. D. Green. Strategies for the systematic sequencing of complex genomes. *Nature Reviews Genetics*, 2(8):573–583, August 2001.

[21] J. Haldane. The comparative genetics of color in rodents and carnivora. *Biol. Rev. Camb. Philos. Soc*, 2:199–212, 1927.

[22] The *Arabidopsis* Genome Initiative. Analysis of the genome sequence of the flowering plant *Arabidopsis thaliana*. *Nature*, 408:796–815, 2000.

[23] Y. H. Ju and Y. T. Chuan. An efficient algorithm for finding a maximum weight 2-independent set on interval graphs. *Information Processing Letters*, 43(5):229–235, 1992.

[24] J. Kim, L. Gordon, P. Dehal, et al. Homology-driven assembly of a sequence-ready mouse bac contig map spanning regions related to the 46-mb gene-rich euchromatic segments of human chromosome 19. *Genomics*, 74:129–141, 2001.

[25] A. Kwitek, P. Tonellato, et al. Automated construction of high-density comparative maps between rat, human and mouse. *Genome Research*, 11:1935–1943, 2001.

[26] J. Messing. The universal primers and the shotgun dna sequencing method. *Methods Mol. Biol.*, 167:13–31, 2001.

[27] R. J. Mural et al. A comparison of whole-genome shotgun-derived mouse chromosome 16 and the human genome. *Science*, 296:1661–1671, May 2002.

[28] E. W. Myers et al. A whole-genome assembly of *Drosophila*. *Science*, 287:2196–2204, 2000.

[29] W. Nash and S. O'Brien. Conserved regions of homologous g-banded chromosomes between orders in mammalian evolution: Carnivores and primates. *Proc. Nat. Acad. Sci. USA*, 79:6631–6635, 1982.

[30] S. Nilsson, K. Helou, A. Walentinsson, C. Szpirer, O. Nerman, and F. Ståhl. Rat-mouse and rat-human comparative maps based on gene homology and high-resolution zoo-fish. *Genomics*, 74:287–298, 2001.

[31] R. J. Oakley, M. L. Watson, and M. F. Seldin. Construction of a physical map on mouse and human chromosome 1: Comparison of 13 mb of mouse and 11 mb of human dna. *Human Molecular Genetics*, 1:616–620, 1992.

[32] S.J. O'Brien et al. The promise of comparative genomics in mammals. *Science*, 286:458–481, October 1999.

[33] M. V. Olson et al. Random-clone strategy for genomic restriction mapping in yeast. *Proc. Nat. Acad. Sci. USA*, 83:7826–7830, 1986.

[34] P. Onyango, W. Miller, J. Lehoczky, C. Leung, B. Birren, S. Wheelan, K. Dewar, and A. P. Feinberg. Sequence and comparative analysis of the mouse 1 megabase region orthologous to the human 11p15 imprinted domain. *Genome Research*, 10:1697–1710, 2000.

[35] C. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *J. Computer System Sciences*, 43:425–440, 1991.

[36] A. H. Paterson, T. H. Lan, K. P. Reischmann, et al. Toward a unified genetic map of higher plants, transcending the monocot-dicot divergence. *Nature Genetics*, 14(4):380–382, December 1996.

[37] M. T. Pletcher, T. Wiltshire, D. R. Cabin, M. Villanueva, and R. H. Reeves. Use of comparative physical and sequence mapping to annotate mouse chromosome 16 and human chromosome 21. *Genomics*, 74:45–54, 2001.

[38] L. Riles et al. Physical maps of the six smallest chromosomes of *Saccharomyces cerevisiae* at a resolution of 2.6 kilobase pairs. *Genetics*, 134:81–150, 1993.

[39] F. Sanger, A. R. Coulson, G. F. Hong, D. F. Hill, and G. B. Petersen. Nucleotide sequence of the bacteriophage lambda dna. *J. Molecular Biology*, 162:729–773, 1982.

[40] F. Sanger, S. Nicklen, and A. R. Coulson. Dna sequencing with chain-terminating inhibitors. *Proc. Nat. Acad. Sci. USA*, 74(12):5463–5468, 1977.

[41] J. Sawyer and J. C. Hozier. High resolution of mouse chromosomes: Banding conservation between man and mouse. *Science*, 232:1632–1635, 1986.

[42] H. Scherthan, T. Cremer, et al. Comparative chromosome painting discloses homologous segments in distantly related mammals. *Nature Genetics*, 6:342–347, 1994.

[43] S. Schwartz, Z. Zhang, K. A. Frazer, A. Smit, C. Riemer, J. Bouck, R. Gibbs, R. Hardison, and W. Miller. Pipmaker – a web server for aligning two genomic dna sequences. *Genome Research*, 10:577–586, 2000.

[44] L. Stubbs, E. M. Rinchik, et al. Clustering of six human 11p15 gene homologs within a 500 kb interval of proximal mouse chromosome 7. *Genomics*, 24:324–332, 1994.

[45] J. W. Thomas, T. J. Summers, et al. Comparative genome mapping in the sequence-based era: Early experience with human chromosome 7. *Genome Research*, 10:624–633, 2000.

[46] J. C. Venter et al. The sequence of the human genome. *Science*, 291:1304–1351, February 2001.

[47] J. Weber and G. Myers. Whole genome shotgun sequencing. *Genome Research*, 7:401–409, 1997.

[48] J. Weinberg and R. Stanyon. Chromosome painting in mammals as an approach to comparative genomics. *Curr. Opin. Genet. Dev.*, 5:792–797, 1995.

[49] Z. Zhang, S. Schwartz, L. Wagner, and W. Miller. A greedy algorithm for aligning dna sequences. *J. Computational Biology*, 7:203–214, 2000.

Vita

Vamsi Veeramachaneni was born in Warangal, India, on November 28th, 1971. He obtained a B.Tech in Computer Science and Engineering from the Indian Institute of Technology, Kanpur, in 1993. After completing a M.S. in CSE at Iowa State University in 1995, he enrolled in the PhD program at Penn State University. He worked as a teaching assistant, graphics programmer, bioinformatics programmer and research assistant till he defended his thesis in August 2002.

He is currently employed as a Research Associate in the Department of Biology. Vamsi and his wife, Kavitha, plan to return to India in the near future to pursue careers in research.