Separating repeats in DNA sequence assembly*

(Extended abstract)

John Kececioglu† Jun Yu‡

Abstract

One of the key open problems in large-scale DNA sequence assembly is the correct reconstruction of sequences that contain repeats. A long repeat can confound a sequence assembler into falsely overlaying fragments that sample its copies, effectively compressing out the repeat in the reconstructed sequence. We call the task of correcting this compression by separating the overlaid fragments into the distinct copies they sample, the *repeat separation problem*. We present a rigorous formulation of repeat separation in the general setting without prior knowledge of consensus sequences of repeats or their number of copies. Our formulation decomposes the task into a series of four subproblems, and we design probabilistic tests or combinatorial algorithms that solve each subproblem. The core subproblem separates repeats using the so-called k-median problem in combinatorial optimization, which we solve using integer linear-programming. Experiments with an implementation show we can separate fragments that are overlaid at 10 times the coverage with very few mistakes in a few seconds of computation, even when the sequencing error rate and the error rate between copies are identical. To our knowledge this is the first rigorous and fully general approach to separating repeats that directly addresses the problem.

Keywords Computational biology, shotgun sequencing, disambiguating repeats, k-median problem

1 Introduction

Now that DNA sequencing has progressed to sequencing entire genomes of complex organisms such as man, the problem of correctly dealing with repeats in such sequences is receiving more and more attention. While approaches to sequence assembly that work correctly in the absence of repeats have been known for some time [17, 13, 6, 15], they tend to compress sequences that contain long repeats. Several interesting alternate approaches have been suggested, based on trying to find an assembly whose coverage does not deviate too far in distribution [16], or by trying to accomodate constraints on the orientation and distance between pairs of fragments sampled from the ends of longer clones [19, 2]. Finding an assembly whose depth is uniformly close to the expected coverage does not prevent, for instance, mixing fragments from different copies of a repeat; for that analysis of the fragment sequences is required. Accomodating constraints on end-pairs of fragments can effectively extend the size of the unit that is read from the underlying sequence, and hence can span a repeat by sampling disambiguating context, but even here a sufficiently long repeat, for instance formed from tandem copies of shorter units, will ultimately confound such an approach.

Indeed in the worst case one can show that it is impossible to correctly reconstruct a sequence that contains exact copies of a repeat. Fortunately in genomic DNA long repeats are almost never exact copies, but usually contain small differences distributed throughout. Such differences between copies, which are usually called *distinguishing base sites*, permit in principle the correct assembly of sequences even with arbitrarily long repeats. While it has been recognized by researchers in several public forums that analysis of distinguishing base sites is necessary to correctly separate repeats, we are not aware of any rigorous and fully automatic method for carrying out such an analysis in the literature on sequence assembly algorithms [10, 8, 13, 19, 16, 5, 7, 15, 3, 4, 20, 2].

We present a four-phase approach to separating repeats which works with any assembler that follows the standard three-phase decomposition [17] into

(1) constructing a graph whose edges represent how fragments *overlap*,

(2) selecting edges to form a fragment *layout*, and

(3) multiple alignment of the layout to recover a sequence by *consensus*.

Moreover our approach is fully *general*: it does not require prior knowledge of consensus sequences for repeats, or their number of copies. The only sequencing parameters that are needed are:

*To appear in Proceedings of the 5th ACM *Conference on Computational Molecular Biology* (RECOMB 2001). Research supported by U.S. National Science Foundation CAREER Award DBI-9722339 and U.S. National Science Foundation Grant DBI-9872649.

†Corresponding author. Department of Computer Science, The University of Arizona, Tucson, AZ 85721-0077, USA. Email: kece@cs.arizona.edu

‡Department of Computer Science, The University of Georgia, Athens, GA 30602-7404, USA. Email: jun@cs.uga.edu

- the coverage depth r of the sequencing project,
- an upper bound ϵ on the sequencing error rate, and
- a lower bound δ on the error rate between copies.

Furthermore our algorithms are fully *automatic*, and contain no ad hoc internal parameters. All parameters internal to the algorithms have natural interpretations as confidence bounds on well-defined probability events.

Informally we take repeat separation to be the following task. The *input* is a candidate assembly described by a layout of the fragments. The *output* is a set of edges in the overlap graph that are inferred to be false overlaps, due to them overlapping fragments that sample different copies of a repeat. We assume that along with the layout there is a procedure that, given any region in the layout delimited by the left and right ends of two fragments, returns a multiple alignment of the fragments that span the region.

A procedure for repeat separation with this input-output behavior can be combined with a three-phase assembler as follows. Iterate the layout phase by computing a candidate layout, running the separation procedure on the layout and removing from the graph all edges identified as false overlaps, and repeat, stopping once no false edges are found with respect to the current layout. This idea of solving a problem with unknown error in the input (here the false overlaps in the overlap graph) by computing an initial solution, using the solution to identify and remove some of the error, recomputing a solution on the edited input, and repeating, is not new in computational biology, and has been suggested for instance in chromosome physical mapping.

Modeling the task of repeat separation as above has several advantages. It isolates repeat separation from the task of computing layouts, so a procedure for repeat separation can be combined with any three-phase assembler. Moreover this modularization simplifies the layout algorithms, since a layout algorithm that is correct in the simpler setting of assembly in the absence of repeats can be combined with a procedure that solves the repeat separation problem to yield an assembler that is correct in the presence of repeats.

A potential disadvantage is that several iterations of the layout phase may be necessary to obtain a layout that does not compress repeats. The layout phase, however, is the least computationally-intensive phase in practice [10], usually at least an order of magnitude faster than the overlap phase. Furthermore, our procedure for repeat separation is carefully designed to be fast (as demonstrated in Section 6), so repeated iteration should not significantly increase the total time for assembly.

Overview Our approach to repeat separation decomposes the task into four subproblems:

(1) *locating* the positions in the layout of possible compression due to repeats,

(2) *bounding* a position of potential compression by a window in the multiple alignment of the fragments,

(3) *identifying* the columns in the alignment window that represent distinguishing base sites, and

(4) *separating* the fragments in the window into classes that represent the distinct copies of the compressed repeat.

In Sections 2 through 5 of the paper, we formalize each of these subproblems using the language of probability and combinatorial optimization, and present algorithms or tests that solve them. Section 6 then presents computational results with an implementation of this approach.

2 Locating potential compression

Given a candidate layout, we first locate positions in the layout that potentially compress repeats. We assume the layout is described by the position of the left and right ends of the fragment intervals in each contig. After separately sorting the list of left positions and right positions into increasing order for a given contig, we walk these two lists in merged order to determine the elementary intervals of the layout, i.e. the layout intervals in which no fragment starts or ends, while maintaining a count of the coverage depth in each such interval. Each elementary interval is considered to be a location in the layout.

We identify locations of potential compression by the presence of unexpectedly high coverage, which suggests that fragments from different copies have been stacked by the layout. Of course unusually high coverage can also be due to ordinary variation in coverage depth, so each such location is analyzed by further phases that examine the sequence content of the stacked fragments. Filtering layout locations before performing further analysis reduces the running time significantly, since analyzing the location will involve computing a multiple alignment of the stacked fragments and solving an optimization problem to partition them into copy classes, both of which are potentially expensive.

We note that repeats may also be compressed at locations of typical coverage. To not miss such compression, we suggest that once a final layout L has been found with no locations of unexpectedly high coverage, the later phases of repeat separation should be run at all locations in L.

It should also be pointed out that in actuality the processes of locating potential compression in phase (1), and analyzing such locations in phases (2) through (4), are interleaved. On identifying a location of high coverage in phase (1), phases (2) through (4) are immediately performed, and their output is used to determine the next location considered by phase (1).

A location of coverage depth d is classified as having unexpectedly high coverage if the probability of having depth at least d is sufficiently small. We denote the *cumulative binomial distribution function* by

$$B(p, m, n) \ := \sum_{m \le i \le n} \binom{n}{i} p^i (1-p)^{n-i}. \qquad (1)$$

The probability of observing a given coverage depth under a natural sequencing model can be expressed in terms of this function. We denote the total number of fragments in the assembly by n, the length of the genome by g, and the length of a fragment by ℓ.

Lemma 1 *Suppose the left ends of fragments are uniformly distributed across the genome, that fragments are labeled by distinct names, and that all fragments have the same length. Then the probability of having depth at least d at a fixed point in the layout is $B\left(\frac{\ell}{g-\ell+1}, d, n\right)$.* □

Given a *compression confidence-threshold* χ, which is a small probability set by the user, using this lemma we classify a location as having potential compression if

$$B\left(\tfrac{\ell}{g-\ell+1}, d, n\right) \ \le \ \chi. \qquad (2)$$

In practice, we precompute the quantity

$$d^* \ := \ \min_{1 \le d \le n} \left\{ d \ : \ B\left(\tfrac{\ell}{g-\ell+1}, d, n\right) \le \chi \right\}$$

by unbounded binary search on d. (Note that n, g, and ℓ are constants.) At a location of depth d we then simply test whether $d \ge d^*$ in $O(1)$ time. This is equivalent to testing inequality (2), since $B(\cdot, d, \cdot)$ is a decreasing function of d.

To evaluate B in the preprocessing we do not use equation (1), but instead express it in terms of the incomplete beta function (see [18, pp. 178–180]), which allows us to evaluate $B(p, m, n)$ in worst-case time $O(\sqrt{\max\{m, n-m\}}) = O(\sqrt{n})$. Finding d^* by unbounded binary search then takes $O(\sqrt{n} \log d^*)$ time. This is dominated by the time to initially sort the endpositions of the n fragments, which takes $O(n \log n)$ time.

To summarize, given threshold χ from the user, after $O(n \log n)$ time preprocessing we test whether a particular layout location has potential compression in $O(1)$ time.

3 Bounding a location by a window

Given a location of potential compression, we need to analyze the sequences stacked at that location to separate the fragments into copies. (If the fragments all sample a common position in the genome, we view them as sampling one copy.) To analyze the sequences and separate them on the basis of sites that distinguish the copies, we need a multiple alignment of the fragments, whose boundaries must be specified. The most natural boundaries of the alignment are the endpoints of the elementary interval at the given location in the layout. The alignment window specified by these boundaries is likely to be rather narrow, however, and to separate the fragments we need a window that is wide enough to contain several distinguishing base sites among its columns. On the other hand, our separation procedure, described in Section 5, uses Hamming distances between fragments that are computed over the rows of the alignment, and these distances will be biased if we consider fragments whose row does not span the entire window, so we only include in a window fragments that span it. This has the consequence that widening a window will reduce the number of fragments it contains, which creates the following tension. A deep window containing many fragments will be narrow and is unlikely to have enough distinguishing base sites, while a wide window containing many such sites will be shallow and not allow us to separate many fragments. What is the best window that both separates many fragments and has sufficiently many distinguishing base sites?

We resolve this problem in the following way. Using probability, we characterize when a window of a given width and depth is likely to be sufficiently wide to contain enough recognizable distinguishing base sites that we can separate nearly all the copies sampled by its fragments. In other words, we design a probabilistic test for when a window of a given depth is sufficiently wide. If a window passes the test we say it is *feasible*.[1] When examining feasible windows, it suffices to consider a widest window of any given depth. Let us call a widest window of a given depth a *representative* window for that depth. The *optimal window* at a layout location is a representative window that is feasible and has greatest depth.

To find an optimal window we solve the combinatorial problem of computing a representative window for a given depth. At a location of coverage depth d, we then successively compute a representative window of depth d, $d-1$, $d-2$, ..., until we find one that is feasible.

We now present our criterion for deciding when a column of the alignment represents a distinguishing base site. We then develop our probabilistic test for feasibility in terms of bounds on the number of true positive and false positive sites in a window, and finally describe our algorithm for the representative window problem.

3.1 Characterizing distinguishing-base-sites

A distinguishing base site is a position at which the consensus sequences of the copies differ. If the coverage is high, such a site will appear in an alignment as a column with significant error. The error, however, will be systematic, as opposed to random sequencing error, in that at all such sites the fragments that sample a common copy will tend to agree, while the fragments that sample different copies will tend to disagree. In other words, the errors at such sites will be correlated.

We call columns with correlated errors *separating columns*, and use the following criterion for identifying them.[2] A *candidate* for a separating column in a window of depth d is a column C in which a pair of rows agree on a character that has frequency at most $\lfloor d/2 \rfloor$ in C. (Such a character is considered to be an *error*.) Column C is *supported* by another column D if both C and D are candidates on the same pair of rows. Finally, column C is a *separating column* if it is supported by at least $t-1$ other columns in the window on the same pair of rows, where $t \ge 1$ is a parameter. (We describe how t is chosen in the next subsection.) Stated another way, columns C_1, C_2, ..., C_t in a window are separating columns if there is a fixed pair of rows (i, j) such that in all these columns the characters at rows i and j agree and occur in the minority. We call parameter t the *support* of the separating column; it controls the extent to which a pair of rows have a correlated error across columns.

Given this definition of a separating column, we determine whether a window of width w and depth d is sufficiently wide in two steps. We first choose t as function of w and d so that the probability of a *false positive* (i.e. finding a separating column that does not correspond to a distinguishing base site) is acceptably low. We then declare a window to be sufficiently wide if the probability of a *true positive* (i.e. finding a separating column that does correspond to a distinguishing base site) is acceptably high. We now describe how we determine these false positive and true positive probabilities.

3.2 Upper-bounding false positives

Let C_1, C_2, ..., C_t be a group of separating columns that support each other. Informally, we consider each column in the group to be a false positive if none of the C_i are distinguishing base sites. (If at least one of the C_i is a distinguishing base site, then declaring the set to be separating columns correctly identifies a distinguishing base site, while

[1] We use a probabilistic feasibility test because, as we will see, actually recognizing the distinguishing base sites in a window is relatively expensive, and we want the feasibility test to be cheap since we may potentially invoke it on many windows.

[2] Separating columns are intended to represent distinguishing base sites. We use a different term, since they might not correspond to such sites.

including only a small number of erroneous columns that happen to reinforce the base site.)

In formalizing this we use the following probability model. We assume each character in the multiple alignment independently is a *sequencing error* with probability ϵ, where ϵ is the sequencing error rate. We further assume that each character in the consensus sequence for a copy, induced by the rows of multiple alignment that correspond to fragments that sample the copy, independently is a *copy error* with probability $\delta/2$, where δ is the copy error rate.[3] Intuitively, each copy is formed by copying a common sequence with error rate $\delta/2$ (so the error rate between two copies is δ), and then each fragment is formed by copying one of the copies with error rate ϵ.

In this model, columns C_1, ..., C_t form a *false positive* if

- there exists a pair (i, j) of rows such that in each of the t columns, the characters at these two rows are identical and are errors (so the columns form a positive event), but
- none of these $2t$ errors are copy errors (so it is a false event).

The first condition states that the C_i are separating columns, while the second states that none of them correspond to a distinguishing base site.

Determining the probability of a false positive event under this model is complex, and even if it could be worked out it is unlikely it could be efficiently evaluated. To make the analysis tractable, we further restrict the model by assuming that each row in a window of width w has exactly the expected number of errors, namely $\lceil \epsilon w \rceil$. Working out the exact false positive probability is difficult even in this model. Fortunately, our objective is simply to ensure that the false positive probability is small, and for this an upper bound suffices. If the upper bound is small, the exact probability is small.

To obtain an upper bound on the false positive probability in this restricted model, we analyze a necessary condition for a false positive event to occur in a fixed pair of rows. This yields an upper bound on the probability of a false positive, conditioned on a pair of rows. To remove the conditioning we sum over all pairs, which further overestimates the true probability as the conditioned events are not disjoint. The necessary condition on a false positive event that we analyze is:

- after distributing $\lceil \epsilon w \rceil$ errors in both rows of the fixed pair, there are at least t columns in common at which sequencing errors occur, and
- at least t columns out of a possible $\lceil \epsilon w \rceil$ common columns have identical errors in the two rows and do not contain copy errors.

We omit the full details in this abstract, and only state the final result. (Proofs of many of the lemmas are given in [21].)

Lemma 2 *Let t be the support for a separating column. The probability that a false-positive separating column occurs in a window of width w and depth d in the above model is upper-bounded by*

$$f(t, w, d) \;:=\; \binom{d}{2}\left(1 - \frac{1}{\binom{w}{k}} \sum_{0 \le i < t} \binom{k}{i}\binom{w-k}{k-i}\right) \cdot B\left(\tfrac{1}{4}(1-\tfrac{\delta}{2})^c, t, k\right),$$

where ϵ is the sequencing error rate, δ is the copy error rate, r is the genome coverage, B is the cumulative binomial distribution function, $k := \lceil \epsilon w \rceil$ is the number of sequencing errors in a row, and $c := \max\{d/r, 1\}$ is the estimated number of copies in the window. □

[3] A copy error is what creates a distinguishing base site.

For a given *support confidence-threshold* σ provided by the user, the support that we use for identifying separating-columns in the window is then

$$t^* \;:=\; \min_{t \ge 1}\big\{t \,:\, f(t, w, d) \le \sigma\big\}.$$

In other words, t^* is the smallest support that guarantees that the probability of a false positive is at most σ. We note that $t^* \le k = \lceil \epsilon w \rceil$, since $f(k, w, d) = 0 \le \sigma$.

We evaluate $f(t, w, d)$ as follows. The logarithm of the binomial coefficient $\binom{n}{m}$ can be computed to within machine precision in $O(1)$ time using Lanczos' method for the logarithm of the gamma function (see [18, pp. 167–170]). As mentioned in Section 2, $B(\cdot, \cdot, k)$ can be computed in $O(\sqrt{k})$ worst-case time. So given $f(t-1, w, d)$, we can evaluate $f(t, w, d)$ in $O(\sqrt{k})$ time. Thus we can find t^* by linear search in worst-case time

$$O(t^* \sqrt{k}) \;=\; O(k^{3/2}) \;=\; O(\lceil \epsilon w \rceil^{3/2}),$$

which is extremely fast. In practice, usually $2 \le t^* \le 4$.

As an aside, the reason we use such an involved method to pick t^* is that there does not appear to be one value of t^* that is appropriate for windows of all shapes, and we want a fully automatic approach with no ad hoc parameters. Furthermore, we have to be careful in selecting t^* since too low a value will introduce false positives, which as we will see in Section 5 will cause us to overestimate the number of copies and hence remove true overlaps from the overlap graph. Selecting too high a value will introduce false negatives, which is not as serious, but as we will see in the next subsection this tends to make the optimal window shallower than necessary. Having a shallower window decreases the number of fragments separated at a given location, which will tend to increase the number of iterations of the layout phase.

3.3 Lower-bounding true positives

Determining the probability of a true positive in the above model is also difficult. Our objective however is to determine whether the probability of a true positive is acceptably high for a given window. Hence a lower bound on the probability suffices.

To obtain a lower bound we study the probability of a true positive for a fixed copy C, conditioned on exactly $i \ge t^*$ copy errors occuring in C. We determine this probability exactly for $i = t^*$, and show this lower bounds the probability when $t^* < i \le \lfloor w/2 \rfloor$. Finally we remove the conditioning by summing over all i in this range, where nearly all the probability is concentrated.

To show the lower bound for $i > t^*$ copy errors, we use an interesting argument. Call an *i-configuration* a labeling of i columns as having copy errors, together with a distribution of $\lceil \epsilon w \rceil$ sequencing errors in each row of the window so that some pair of rows are sequencing-error-free on a common set of at least t^* of the i columns. An i-configuration is a particular type of true positive event that carries a lot of the probability mass. In particular, t^*-configurations capture *all* true positive events when exactly t^* copy errors have occurred. While it is difficult to count the number of i-configurations exactly for general i, we show the existence of a one-to-one mapping from the set of t^*-configurations to the set of i-configurations for $t^* < i \leq \lfloor w/2 \rfloor$, which establishes the lower bound. Existence is proved by constructing a bipartite graph on the set of t^*-configurations and i-configurations, and showing that Hall's condition for a complete matching is satisfied by this graph (see [14, p. 283]).

We omit the details in this abstract and state the final result.

Lemma 3 *Let t be the support for a separating column. The probability that a true-positive separating column occurs for a given copy in a window of width w and depth d is lower-bounded by*

$$g(t,w,d) \;:=\; \left(B\!\left(\tfrac{\delta}{2}, t, w\right) \;-\; B\!\left(\tfrac{\delta}{2}, \left\lfloor \tfrac{w}{2} \right\rfloor + 1, w\right) \right) \cdot \left(\frac{\binom{w-t}{k}}{\binom{w}{k}}, 2, \min\{r,d\} \right),$$

where $k := \lceil \epsilon w \rceil$. □

If a window contains a true-positive separating column for each copy, we can in principle separate all its fragments into their copy classes. In our probability model we assume each fragment is labeled by the copy it samples, so extending the analysis from the probability of a true-positive for a fixed copy to the probability of a true-positive in each of c copies is trivial, since errors in the copies are independent. This leads to the following window feasibility test.

For a given *window confidence-threshold* ω and *copy confidence-threshold* κ (which are both small probabilities provided by the user), a $d \times w$ window is *feasible* if

$$B\Big(g(t^*, w, d), \lceil (1-\kappa)c \rceil, c\Big) \;\geq\; 1 - \omega,$$

where $c := \max\{\lceil \frac{d}{r} \rceil, 1\}$ is the estimated number of copies for a given coverage r, and t^* is the support for the window determined in Section 3.2. Stated another way, a window is feasible if the probability of not finding a true-positive separating-column for at most a fraction κ of the copies is at most ω.

Using the same numerical techniques as in Section 3.2, we can evaluate $g(t,w,d)$ in worst-case time $O(\sqrt{w} + \sqrt{d}) = O(\max\{w,d\}^{1/2})$.

To summarize, given confidence thresholds σ, ω, and κ, we can test whether a window is feasible in time

$$O(\lceil \epsilon w \rceil^{3/2} + \max\{w,d\}^{1/2}).$$

We next explain how to quickly find, among the windows that are feasible, one that is optimal.

3.4 Finding an optimal window

Recall that an *optimal* window is a representative window that is feasible and has maximum depth, and that a *representative* window is a widest window of a given depth, where the *width* of a window is the length of the intersection of the layout intervals of its fragments. The endpoints of a fragment's interval are the positions of the start- and end-columns of its row in the multiple alignment.

Let d be the depth of the layout at the location where we want to compute an optimal window. To find an optimal window, we will compute a representative window of depth d, $d-1$, $d-2$, ... until we find one that is feasible. A representative window of depth $d-k$ can be described by the set of k fragments it removes from the layout. To quickly find such a window we cannot afford to examine all $\binom{d}{k}$ subsets for removal. We now show how to find a representative window of depth $d-k$ while examining at most $k+1$ subsets.

Consider sorting the left ends of the fragment intervals in decreasing order, and the right ends in increasing order. For a given depth $d-k$, we call the window that removes the fragments with the k greatest left-ends, the *leftmost* solution. Similarly the window that removes the fragments with the k smallest right-ends is the *rightmost* solution. (Note that the leftmost and rightmost solutions are not necessarily distinct.)

In general, define a *(k,i)-window*, where $0 \leq i \leq k \leq d$, to be the window of depth $d-k$ obtained by

(1) removing the fragments with the $k-i$ greatest left-ends, followed by

(2) removing from the remaining fragments those with the i smallest right-ends.

For example, the $(k,0)$-window is the leftmost solution and the (k,k)-window is the rightmost solution. The following two lemmas on (k,i)-windows yield an efficient algorithm for computing a representative window.

Lemma 4 *The widest window among the $(k,0)$-, $(k,1)$-, ..., (k,k)-windows is a representative window of depth $d-k$.* □

Lemma 5 *Given a representation of the list of (k,i)-windows for k fixed and i varying, we can compute the representation for the list of $(k+1,i)$-windows for k fixed and i varying, in $O(k)$ total time, using $O(k)$ space.* □

Applying Lemma 5 for $k = 0, 1, \ldots$ yields an algorithm that finds an optimal window in $O(\ell^2)$ time and $O(\ell)$ space, not counting the time for the feasibility test, where ℓ is the first k at which a feasible window is found. Since we only test one window for feasibility at each k, counting these tests adds $O(\ell \max\{d,w\}^{1/2})$ time, where w is the width of the optimal window. In practice ℓ, the number of fragments we have to remove to obtain a feasible window, is small.

To summarize, given a location in the layout of depth d, we can find the optimal window bounding the location in time

$$O(k^2 + k \max\{d,w\}^{1/2})$$

using $O(k)$ space, where $d-k$ and w are the depth and width of the optimal window.

4 Identifying separating-columns

Given a window in the multiple alignment, our next task is to identify its separating columns. As defined in Section 3.1, a separating column with support t in a window of depth d is a column C_1 such that there are $t-1$ other columns C_2, C_3, ..., C_t with the following property: there is pair of rows (i,j) such that in columns C_1, ..., C_t the characters at rows i and j are equal and occur in the column with frequency at most $\lfloor d/2 \rfloor$. We identify all such columns in two passes over the window as follows.

In the first pass over the columns we fill in a table $T(i,j,c)$, which counts the number of columns where character c appears in rows i and j and has frequency at most $\lfloor d/2 \rfloor$. Table T is of size $\Theta(d^2)$ (since for DNA sequences the alphabet is of size 4, which is a constant), and can be computed in $\Theta(wd^2)$ time.

In the second pass we again examine each column C and consider pairs of rows (i, j) in C. Now if rows i and j agree on a character c that appears at most $\lfloor d/2 \rfloor$ times and $T(i, j, c) \geq t$, then C is a separating column. This second pass also takes $O(wd^2)$ time.

In short, given a window of width w and depth d, we can find all separating columns in $O(wd^2)$ time using $O(d^2)$ additional space.

5 Separating fragments into copies

Given the separating columns for the window, which represent the inferred distinguishing base sites of the copies, we would like to partition the fragments in the window into copies based on how they differ at these columns. In the partition, each class represents the set of fragments that sample a different copy of the repeat. To determine the best partition of the fragments into copy classes we need an appropriate objective function on partitions.

Suppose the fragments in the window sample k different copies of a repeat. In a parsimony formulation of the problem, we would want an explanation of the fragments by k copies that involves the fewest number of sequencing errors. Formally, we want a partition of the fragments into k classes P_1, ..., P_k, together with k strings S_1, ..., S_k representing the consensus sequences of the copies, that minimizes $\sum_{1 \leq i \leq k} \sum_{F \in P_i} D(F, S_i)$, where $D(\cdot, \cdot)$ counts the number of errors between two strings.

If we accept the multiple alignment in the window as being accurate, $D(F, S_i)$ can be evaluated for a given partition as follows. In the row of the alignment corresponding to fragment F, count the number of columns at which the character in F does not have maximum frequency among the rows in P_i; this count equals $D(F, S_i)$. Computing $D(\cdot, \cdot)$ this way reduces the problem to partitioning the rows of an alignment into k classes. This eliminates the consensus sequences from the formulation, which makes the problem much simpler, but the resulting objective function on partitions is still awkward to work with.

Notice, however, that if we only evaluate this objective over the separating columns, which represent the columns that distinguish the copies, it is quite likely that in every class P_i of the partition, there is one row that coincides with the consensus sequence at all columns. This is because the separating columns are a sparse set and sequencing errors are infrequent, so with high probability at least one row in each class has no sequencing errors at these columns. Suppose the row for fragment F^* coincides at the separating columns with the consensus sequence for P_i. Then $D(F, S_i)$ on the separating columns is equal to the Hamming distance between the rows for F and F^* at these columns. (The *Hamming distance* between two vectors is the number of positions at which they differ.) Thus our objective function becomes, over the multiple alignment restricted to the separating columns, find a partition of the rows into k classes P_1, ..., P_k that minimizes

$$\sum_{1 \leq i \leq k} \min_{F^* \in P_i} \Big\{ \sum_{F \in P_i} H(F, F^*) \Big\}, \qquad (3)$$

where $H(\cdot, \cdot)$ is the Hamming distance between two rows of the alignment.

This form of the objective is equivalent to the following problem on graphs. A *star* in a graph is a set of edges that all touch a common vertex, called the *center* of the star. (A single vertex is the center of a star with no edges, and a single edge is a star where either endpoint is the center.) A *k-star* is a collection of k vertex-disjoint stars, and a *spanning* k-star is one whose vertex set is the entire graph. In this language, minimizing objective (3) is equivalent to the following: Given a complete, edge-weighted graph G on d vertices, where edge (i, j) is weighted by the Hamming distance between rows i and j of the alignment restricted to its separating columns, find a spanning k-star of minimum total edge-weight. We call this the *k-star problem.* The stars in a solution partition the fragments into k copy classes.

In the combinatorial optimization community the k-star problem is known as the *k-median problem.* It is known to be NP-complete even when the edge weights are a metric (as is the case with Hamming distances), and only fairly recently have constant-factor approximation algorithms been discovered for this situation (see [9] for a recent 6-approximation algorithm). Since we are using the k-star problem to separate fragments into copies, and any overlap between the copy classes of a solution will be removed from the overlap graph when computing a layout, it is critical that we find an optimal solution. We find an optimal spanning k-star by formulating it as an integer linear-programming problem, which we solve by branch-and-bound.

Before describing the branch-and-bound algorithm we make one final remark. Our discussion up to this point has assumed that we know k, the number of copies, but this parameter must also be inferred from the data. To determine k we again invoke the principle of parsimony. We take k to be the smallest value $i \geq 1$ such that the copy classes of the optimal spanning i-star contain no separating columns. In other words, we successively solve the 1-star, 2-star, 3-star problems, and so on, until we find a k-star solution for which, when we run the procedure of Section 4 for identifying separating columns within the rows of each of its copy classes, none are found. This k is the fewest number of copies that explains the data without indicating further distinguishing base sites within its copies. Note that if the original window has no separating columns, then $k = 1$.

5.1 Integer-programming formulation

Given a k-star problem on the complete graph K_n with edge weights w_{ij} for each edge $(i, j) \in K_n$, we construct an equivalent integer linear-programming problem as follows.

The integer program has $n^2 + n$ *variables*:

- for each ordered pair (i, j), where i and j are vertices in K_n (possibly with $i = j$), there is a variable x_{ij}, and
- for each vertex i in K_n, there is a variable y_i.

The x-variables encode the edges of the k-star, while the y-variables encode the centers of the stars.

The program contains $3n^2 + 3n + 1$ *constraints*:

- for all i and j, $x_{ij} \geq 0$,
- for all i, $y_i \geq 0$,
- for all j, $\sum_{1 \leq i \leq n} x_{ij} \geq 1$,
- for all i and j, $y_i \geq x_{ij}$, and
- $\sum_{1 \leq i \leq n} y_i \leq k$,

where the variables are restricted to integer values.

Table 1 Separating simulated tandem repeats.

		False positives		False negatives		Depth		Time		Nodes	
ϵ	δ	mean	max	mean	max	mean	max	mean	max	mean	max
5.0%	5.0%	0.10%	0.26%	0.03%	0.07%	92	117	13.56	15.23	1.0	1
5.0%	10.0%	5.4%	12.9%	2.2%	8.2%	80	108	5.16	9.33	1.8	5
2.5%	5.0%	6.0%	12.1%	16.0%	35.0%	120	129	9.10	11.22	1.0	1
2.5%	10.0%	0.0%	0.0%	0.0%	0.0%	—	—	0.01	0.01	—	—

The *objective* is to minimize $\sum_{i \neq j} w_{ij} x_{ij}$, where we extend the edge weights to ordered pairs of vertices by $w_{ij} = w_{ji}$.

To describe an optimal k-star it suffices to specify its centers, since its edges may be obtained by assigning each vertex to its nearest center. Given a solution to the integer program, we recover a k-star by taking as its centers those i for which $y_i > 0$. The following lemma states that this correspondence solves the k-star problem.

Lemma 6 *Let x, y be a solution to the above integer program. If all edge weights are non-negative, there is an optimal spanning k-star with the same cost and centers as x, y.* □

Although it may not be immediately obvious, in every optimal solution to the above integer program, all variables have the value 0 or 1 [21].

5.2 Solution by branch-and-bound

We solve the integer program by performing branch-and-bound on its linear programming relaxation. To do this, we retain the inequalities of the integer program but allow the variables to take on real values. If we solve the linear program and its solution is not integral, we choose a fractional variable and branch on two subproblems. In one subproblem the variable is fixed to 0, and in the other subproblem it is fixed to 1. Every fractional solution that we find is transformed into an integral solution by a rounding procedure. During the computation we maintain the best integral solution we have seen so far, which gives a global upper bound on the solution value. Before branching on a fractional solution to the linear program we compare its solution value, which lower-bounds the solution value of the integer program for the subproblem, to the global upper bound; if the fractional solution value is not less than the global upper bound, we prune the subproblem and do not explore it further.

To *round* a fractional solution into an integral one, we collect all fractional y_i and compute

$$a_i := \sum_{j\,:\,x_{ij}>0} w_{ij} \Big/ \big|\{j \,:\, x_{ij} > 0\}\big|.$$

Let ℓ be the number of y-variables that have the value 1. We pick the $k - \ell$ fractional y_i with greatest a_i and assign them the value 1, and assign the remaining y_i the value 0. Finally we round the x_{ij} by assigning every vertex to its nearest center.

To *branch* on a fractional solution, we select the variable whose value is closest to 1/2. We can prove that if the x_{ij} are all integral, the y_i are as well [21], so we only select x-variables. We solve the subproblem with $x_{ij} = 1$ first, and the subproblem with $x_{ij} = 0$ second. When setting $x_{ij} = 1$, we further tighten the constraints of the linear program by forcing $x_{j\ell} = 0$ for all ℓ.

While for most integer-programming problems branch-and-bound approaches are not especially effective, this branch-and-bound algorithm works surprisingly well, usually solving just a few linear programs before finding an optimal integral solution.

6 Computational results

We now present computational results with an implementation of this approach by the second author [21]. The entire code we tested consisted of around 10,200 lines of C, of which roughly 3,200 are the repeat separation implementation. The additional code consisted of a sequence assembler [10] and a linear-programming solver.

The results from our initial experiments are summarized in Table 1. In the experiments we generated a sequence consisting entirely of a tandem repeat by the following process. For a given copy error rate δ and sequencing error rate ϵ, the sequence was constructed by generating a random sequence of length $1,000$ and then making 10 copies, introducing $1000 \cdot \delta/2$ insertion, deletion, and substitution errors into each copy, concatenating the copies, and then randomly sampling the resulting tandem repeat by fragments of length 500 to 10-fold coverage, randomly reverse-complementing the fragments with probability 1/2, and introducing into each fragment $500 \cdot \epsilon$ errors. All confidence thresholds for our separation procedure were fixed at 1%.

We then measured the number of overlaps between fragments that sampled the same copy that were incorrectly removed by our separation procedure divided by the total number of such overlaps that should be removed, which we report as the percentage of *false positives*, and the number of overlaps between fragments that sampled different copies that were not removed by our procedure divided by same denominator as before, which we report as *false negatives*. We also measured the layout depth at the location where separation was performed, the running time of the separation procedure in seconds, and the number of nodes in the branch-and-bound search tree for the k-star problem. The results are averaged over 10 trials and across all layout positions where separation was performed. Measurements were taken on a 128Mb machine with an Athlon 700MHz processor running Redhat Linux 6.0 with the code compiled under `egcs c++ 1.1.2-12` with the `-O2` option.

As can be seen from the table, the approach works quite well. (In the last row of the table, δ was sufficiently large compared to ϵ that the assembler did not overlay fragments from different copies.) For most instances a single linear program suffices to separate the fragments, and the false positive and negative errors are relatively low. In analyzing the data we found that the false positives appear to be due to systematic misalignment of the fragments by the slidingwindow multiple-alignment heuristic used in [10]. While multiple alignment heuristics are beyond the scope of this paper, nevertheless systematically misaligned columns will be interpreted by the repeat separation procedure as separating columns. Such false separating columns cause it to overestimate the number of copies and hence overseparate the fragments, which contributes false positives in the above experiments. We are currently adding an iterative alignment procedure [1, 12] to the code to correct this, and our impression from analyzing the data is that it may reduce the number of false positives to zero. It is interesting that while such misalignment does not affect the consensus sequence for the assembly, greater accuracy is required for correct repeat separation.

We are also currently integrating the repeat separation code into the sequence assembler of [10] to test how well the complete approach, which iterates the layout phase until finding an uncompressed assembly, reconstructs a sequence containing repeats.

7 Conclusion

We have presented a rigorous, general, and fully automatic approach to separating repeats in DNA sequence assembly. The approach does not assume prior knowledge of consensus sequences for repeats or their number of copies. The only parameters it requires are:

- r, the genome coverage,
- ϵ, an upper bound on the sequencing error rate,
- δ, a lower bound on the error rate between copies,

and four internal parameters that are confidence probabilities for well-defined events:

- χ, the compression confidence,
- σ, the support confidence,
- ω, the window confidence, and
- κ, the copy confidence.

Initial experiments indicate that the approach is promising, but that the quality of the multiple alignment is critical to its success. Further experiments, with a better multiple aligner and an integrated system that iterates on candidate layouts, would be necessary to determine whether it yields a solution to the long-standing problem of sequence assembly in the presence of repeats.

Acknowledgements The first author wishes to thank Hans-Peter Lenhof, Knut Reinert, and Ralf Zimmer for discussions on repeats while visiting the Max-Planck-Institut für Informatik and the German National Research Center for Information Technology, and Bob Robinson and Rod Canfield at the University of Georgia for pointers to the corollary of Hall's theorem on matchings.

References

[1] Anson, E.L. and E.W. Myers. "ReAligner: A program for refining DNA sequence multi-alignments." Proceedings of the 1st ACM *Conference on Computational Molecular Biology*, 9–13, 1997.

[2] Anson, E. and G. Myers. "Algorithms for whole genome shotgun sequencing." Proceedings of the 3rd ACM *Conference on Computational Molecular Biology*, 1–9, 1999.

[3] Chen, T. and S. Skiena. "Trie-based data structures for fragment assembly." Proceedings of the 8th *Symposium on Combinatorial Pattern Matching*, 206–223, 1997.

[4] Chen, T. and S. Skiena. "A case study in genome-level fragment assembly." *Bioinformatics* 16, 494–500, 2000.

[5] Green, P. The `phrap` computer program. `http://www.mbt.washington.edu/phrap.docs/phrap.html`, 1999.

[6] Huang, X. "An improved sequence assembly program." *Genomics* 33, 21–31, 1996.

[7] Huang, X. "Performance of the CAP2 sequence assembly program." *Mathematical Support for Molecular Biology*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume 47, American Mathematical Society, Providence, RI, 259–269, 1999.

[8] Idury, R.M. and M.S. Waterman. "A new algorithm for DNA sequence assembly." *Journal of Computational Biology* 2:2, 291–306, 1995.

[9] Jain, K. and V.V. Vazirani. "Primal-dual approximation algorithms for metric facility location and k-median problems." Proceedings of the 40th IEEE *Symposium on Foundations of Computer Science*, 1999.

[10] Kececioglu, J.D. and E.W. Myers. "Combinatorial algorithms for DNA sequence assembly." *Algorithmica* 13:1/2, 7–51, 1995.

[11] Kececioglu, J., M. Li, and J. Tromp. "Reconstructing a DNA sequence from erroneous copies." *Theoretical Computer Science* 185:1, 3–13, 1997.

[12] Kececioglu, J. and W. Zhang. "Aligning alignments." Proceedings of the 9th *Symposium on Combinatorial Pattern Matching*, Springer-Verlag Lecture Notes in Computer Science 1448, 189–208, 1998.

[13] Kosaraju, S.R. and A.L. Delcher. "Large-scale assembly of DNA strings and space-efficient construction of suffix trees." Proceedings of the 27th ACM *Symposium on Theory of Computing*, 169–177, 1995.

[14] Liu, C.L. *Introduction to Combinatorial Mathematics.* McGraw-Hill, New York, 1968.

[15] Meidanis, J. "A simple toolkit for DNA fragment assembly." *Mathematical Support for Molecular Biology*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume 47, American Mathematical Society, Providence, RI, 271–288, 1999.

[16] Myers, E.W. "Toward simplifying and accurately formulating fragment assembly." *Journal of Computational Biology* 2:2, 275–290, 1995.

[17] Peltola, H., H. Söderlund, J. Tarhio, and E. Ukkonen. "Algorithms for some string matching problems arising in molecular genetics." Proceedings of the 9th IFIP *World Computer Congress*, 59–64, 1983.

[18] Press, W.H., B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, New York, 1988.

[19] Sutton, G.G., O. White, M.D. Adams, and A.R. Kerlavage. "TIGR Assembler: A new tool for assembling large shotgun sequencing projects." *Genome Science and Technology* 1, 9–19, 1995.

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

[21] Yu, J. *Separating Repeats in DNA Sequence Assembly.* M.S. thesis, Department of Computer Science, The University of Georgia, 2000.