Comparing Sequence Scaffolds

Gene Myers
Informatics Research, Celera Genomics Corp.,
45 W. Gude Dr., Rockville, MD 20850, USA.
Email: Gene.Myers@celera.com,

Abstract

The DNA sequence assembler we built for the whole genome shotgun assembly of the human genome, utilizes end-reads of inserts to order and orient assembled contigs into scaffolds for which the distances between consecutive contigs are statistically characterized. We consider the problem of comparing two such scaffolds. Applications include comparison of two distinct assemblies for mutual confirmation, and comparison of scaffold assemblies of BACs to determine a whole genome tiling of the BACs. We formalize the problem and develop efficient algorithms for a number of variations of the problem, the essential result being a sparse algorithm that refines gap estimates based on the overlap evidence.

1 Introduction

Most sequence analysis programs make the assumption that the subject sequence or sequences model contiguous stretches, or contigs, of an underlying genome. However, most large scale genome projects generally produce collections of contigs whose order, orientation, and approximate separating distances are known to varying levels of completeness. For example, the current rough draft human genome effort of the Human Genome Project (HGP) [1] has produced a set of partially ordered contigs, of median size 2750bp, covering each of roughly 30,000 BACs clones of average size 150Kbp, that are partially ordered across the genome. Celera's whole genome shotgun approach [2, 3] produces a set of contigs, of median size 9900bp, that are ordered and oriented into scaffolds spanning an average of 1.7Mbp, each of which is mapped and ordered across the genome. An interesting line of inquiry for many investigators would be how to make their tools work on such data.

In this paper we focus on a particular class of problems associated with the scaffold assemblies that result from assembling shotgun data sets for which end-sequence pairing information is available. End-sequence pairs are generated by running sequencing reactions from both ends of a vector insert. For example, in Celera's whole genome protocol, we build libraries of 2Kbp, 10Kbp, and 50Kbp inserts and determine the sequence of 500-600 base pairs from both ends of these inserts. Thus the input to the assembler is *pairs* of DNA reads whose distance from each other is known to be distributed normally with a well-characterized mean and standard deviation. It is well-known by the Lander-Waterman formula [4] that even with the collection of enough data to cover the genome ten times over, there will statistically be many gaps or regions where the genome was not sampled. So the assembler produces a collection of contigs with interspersed gaps. Without the end-pairs, the order and orientation of these contigs would be unknown and the information of little value. However, the end-pairs connect contigs by virtue of having the reads of their pairs in different contigs. This relation orders and orients the contigs into scaffolds for which the distance between consecutive contigs is statistically characterized, with especially tight variance when many pairs provide the linkage. Figure 1 illustrates the idea of a scaffold. The same Lander-Waterman formula implies that scaffolds are expected to span millions of base pairs on average at coverages as low as 5X, and these large scaffolds are easily mapped to a location on the genome via any coarse-grained genetic map such as a low-density STS map [5].

The problem of comparing such scaffolds arose in several practical contexts within our work on whole genome sequencing. Through successive refinements of the assembly algorithms, we found ourselves with different assembly versions, for which we wanted to understand the differences, if any. Moreover, there was often independently determined sequence data and we wanted to compare our scaffolds against such data for the purpose of validating our techniques. Finally, in the case of the human genome, we were able to use our end-reads to build scaffolds of the light shotgun data produced by the public HGP effort on a BAC clone basis. The public tiling of the BACs, inferred by restriction digest length fingerprints, was not particularly reliable. With our scaffold assemblies fully ordering the data and further completing the sequence of each BAC, we proceeded to produce our own tiling based on observing the overlap in sequence between scaffolds.

What makes the scaffold overlap problem interesting, is the requirement that the contigs of the scaffolds overlap in a way that is consistent with their inter-contig spacing. This is particularly subtle when contigs from the two scaffolds are interleaved as in Figure 1. The paper focuses first on formulating the problem and building up to a sparse variation of it. Then the two critical subproblems of the sparse problem are solved and the paper concludes with performance results.

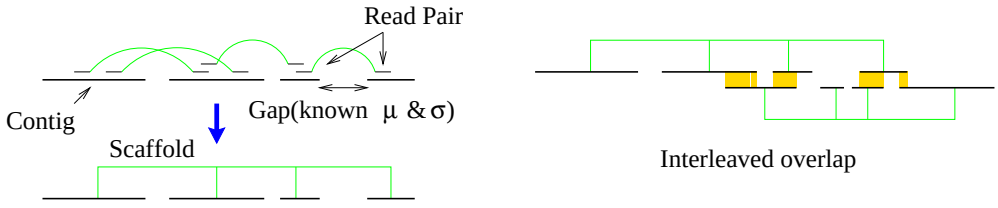

Figure 1: Scaffolds and Scaffold Overlaps.

2 Scaffolds and the Comparison Problems

Formally, a scaffold A consists of n_A contig sequences $A.c_1$, $A.c_2$, ... $A.c_{n_A}$ over alphabet Σ where the length of each gap, say between $A.c_i$ and $A.c_{i+1}$, is assumed to be normally distributed with mean $A.\mu_i$ and standard deviation $A.\sigma_i$. For a given scaffold A, let its *relaxed* sequence be the string $A.c_1\ x^{A.\mu_1}\ A.c_2\ \ldots\ x^{A.\mu_{n_A-1}}\ A.c_{n_A}$ where x is a special wild-card symbol not in Σ and x^k is the string consisting of k $x's$. That is, the relaxed sequence of A is the concatenation of its contigs separated by runs of wild-card symbols whose lengths are the expected lengths of the relevant gaps. Similarly, let the set of Δ-*feasible* sequences for A be the set of strings $A.c_1 x^{\eta_1} A.c_2 \ldots x^{\eta_{n_A-1}} A.c_{n_A}$ where $\eta_i \in [\mu_i \ominus \Delta\sigma_i, \mu_i + \Delta\sigma_i]$ for all i, where $p \ominus q = \max\{0, p-q\}$. That is, the set of Δ-feasible sequences for A is the set of all sequences consisting of the concatenation of its contigs separated by runs of wild-card symbols that can vary by Δ standard deviations from the expected mean length *subject to the restriction that a gap cannot be "negative"*. This last restriction is natural in that a negative gap would generally imply overlap between the adjacent contigs that should have been detected in the construction of the scaffold. At the end of the paper we treat the general case of negative gaps. To recapitulate, the set of feasible strings represent all possible contiguous stretches of sequence underlying the scaffold wherein any gap can be stretched or compressed by at most Δ standard deviations.

For a given pair of scaffolds A and B, the first question to ask is what is the best alignment between them given a threshold Δ on how much the gaps can be adjusted. As for other sequence comparison problems there are the global alignment, the overlap alignment, and the local alignment variations that basically involve exercises in adjusting the boundary conditions of a core algorithmic solution [6]. For our purposes the most relevant version to our intended set of applications is the overlap variation, wherein a prefix and suffix of the sequences may remain unaligned at no charge. We focus on this case and leave the remaining variations as an exercise.

If α and β are strings, then let $\delta(\alpha, \beta)$ be some overlap similarity measure between them, subject to the restrictions that (1) x is assumed to match any symbol including itself at no cost, and (2) every x must be aligned to some character (i.e. it cannot be inserted or deleted). The motivations behind these conditions are (1) an unknown symbol in a gap of a scaffold should match any known symbol of the other scaffold, and (2) the length of a gap is controlled by the number of x's in the run modeling it, and therefore, one should not be able to further stretch it by leaving some of these symbols unaligned.

> *Best Overlap Problem*: Given scaffolds A and B, overlap similarity measure δ, and threshold Δ, compute $\delta_\Delta(A, B) = \max\ \{\ \delta(\alpha, \beta)\ :$ where α and β are Δ-feasible sequences for A and B, respectively $\}$. Moreover, give specific instances of α and β realizing this value and an alignment between them.

Given $\delta_\Delta(A, B)$ there then arises the problem of finding instances α and β realizing this value that stretch their gaps as little as possible. While there are many measures of stretch, the one we study here is related to the physics of spring forces: we seek to minimize the sum of the squares of the deviation from the relaxed state for each gap. This choice minimizes the potential energy of a set of a springs spanning each gap.

> *Optimum Stretch Problem*: Over all α and β for which $\delta(\alpha, \beta) = \delta_\Delta(A, B)$, find the pair (α, β) for which $\Sigma_{i=1}^{n_A-1}(\alpha.\eta_i - A.\mu_i)^2 + \Sigma_{j=1}^{n_B-1}(\beta.\eta_j - B.\mu_j)^2$ is minimal.

Both of these formulations of the problem are easily solvable in $O(MN)$ time where $M = \Sigma_{i=1}^{n_A}|A.c_i| + \Sigma_{i=1}^{n_A-1}(\mu_i + \Delta\sigma_i)$ and N is similarly the length of the longest feasible sequence for scaffold B. To see this observe that the set of feasible strings of A and B can be modeled by regular expressions of length $O(M)$ and $O(N)$. Moreover the finite automaton for these regular expressions are also of the same order and are acyclic graphs. It has long been known that two such regular expressions can be compared in $O(MN)$ time [7] and this solves the best overlap problem. The optimum stretch problem is then solved by simply determining the least stretched alignment over all the optimal paths in the underlying edit graph used to compare the regular expressions.

The typical relaxed size of an assembly scaffold is 25Kbp to 15Mbp, and contigs are at least on the order of thousands of base pairs, if not tens of thousands of base pairs. Moreover, the sequence of the scaffolds is typically 99% or more accurate, so that one expects near-identity matches of significant length between contigs of scaffolds that overlap or correspond to the same sequence. Under these input conditions, the former algorithmic methods can be rendered more practical by moving to a sparse version of the problem where all possible alignments between contigs of the two scaffolds that are above a given quality are computed efficiently in a preprocessing step by a program such as BLAST [8]. Formally, one can assume a set of contig overlap alignments $\Upsilon = \{\tau_1, \tau_2, \ldots \tau_t\}$ where for an overlap τ_i, one is given that the overlap is between contigs $A.c_{\tau_i.a}$ and $B.c_{\tau_i.b}$, has score $\tau_i.score$, and has unaligned prefixes and suffixes specified by signed numbers $\tau_i.beg$ and $\tau_i.end$. For the last two quantities we let $n \geq 0$ imply that the unaligned portion is the first(last) n symbols of the A-contig, and $n < 0$ implies that the unaligned portion is the first(last) $-n$ symbols of the B-contig.

In this sparse variation, the problem becomes one of finding alignments properly composed of a subset of the alignments in Υ. To this end an alignment between two feasible

strings α and β is *valid with respect to* Υ if all aligned segments of non-x symbols are in Υ (and continue to meet the earlier restriction on δ, such as no unaligned x symbols). But perhaps it is simpler to understand this definition in terms of the dynamic programming matrix of α versus β. Term the rectangular submatrix consisting of the cells corresponding to the comparison between an A- and a B-contig, a *block*. Term the rectangular regions for all of β versus an A-gap, a *gap pillar*, and the rectangular regions for all of α versus a B-gap, a *gap beam*. The total matrix is thus partitioned into $n_A n_B$ blocks and a *gap grid* consisting of the union of $n_A - 1$ gap pillars and $n_B - 1$ gap beams. The alignments in Υ are paths from the upper or left border to the lower or right border of the relevant block. An alignment between α and β is valid if it is (a) an alignment from Υ whenever it passes through a block, and (a) a straight diagonal line when it is within the gap grid. Figure 2 illustrates.

For the sparse variation, the problems thus become to find the optimal and least stretched optimal alignment over the space of all those that are valid with respect to Υ. One should note immediately, that it is possible for there to be no valid overlap alignments, a possibility that did not exist for the basic versions of the problem. As for other sparse algorithms, we can convert the problem into one of finding an optimal path through a *stretch graph*. There is one vertex in this graph for each overlap alignment in Υ. There is a directed edge from one vertex, τ, to another, π, if and only if it is possible to connect the end point of τ's alignment to the start point of π's alignment with a diagonal line not passing through a block in the dynamic programming matrix for some pair of feasible strings for A and B. In addition there is a special source vertex, Θ, and special sink vertex, Φ with edges as follows. There is an edge from Θ to a vertex τ if either (a) the start of τ's alignment is at the upper or left border of the dynamic programming matrix, or (b) there is a diagonal line not passing through a block from the upper or left boundary to the start of τ's alignment in the dynamic programming matrix for some pair of feasible strings for A and B. The edges to Φ are similarly constructed where one considers the end of an alignment and the lower and right boundaries.

Within this stretch graph, a path from Θ to Φ represents a valid overlap alignment. This is not immediate as one must verify that there are two specific feasible strings α and β implying the existence of all the edges in the path simultaneously, i.e. one can simultaneously adjust all the gaps so as to imply diagonal, non-block connections between all block alignments used. Consider path $\tau_1 \rightarrow \tau_2 \rightarrow \ldots \rightarrow \tau_n$. Observe that the existence of the edge from τ_i to τ_{i+1} involves properly adjusting the gaps $\tau_i.a$ through $\tau_{i+1}.a - 1$ in the A-scaffold and the gaps $\tau_i.b$ through $\tau_{i+1}.b - 1$ in the B-scaffold. But since $\tau_i.a \leq \tau_{i+1}.a$ and $\tau_i.b \leq \tau_{i+1}.b$ for all i, it follows that the gaps influencing each edge on the path are disjoint, and therefore one can simultaneously adjust gaps to insure the existence of all edges.

It is thus the case that we have mapped the sparse, optimal overlap problem into one of finding a maximum weight path in the acyclic stretch graph, where the weight of each vertex is the δ-score of the block overlap it models. The time for doing so is proportional to the number of edges in the graph which is certainly $O(MN)$ but in most of the scenario's we see in practice it is $O(|\Upsilon|)$. Thus the remaining problem is to devise an efficient procedure for determining the edges of this graph. Moreover, since the gaps relevant to each edge on a path are independent, the optimal stretch problem is easily solved if one additionally determines the minimum stretch possibility for each edge. We now delve into these essential subproblems in the following two sections.

3 The Edge Determination Algorithm

The problem we face in this section is how to determine if two overlap alignments in Υ can be connected by a diagonal line that lies strictly within the gap grid for some choice of the relevant gaps separating the two blocks. The problem has some of the flavor of the guard posting problem seen in computational geometry [9], the interesting twist here being the unusual model/concept of visibility/reachability.

Imperative to a formal treatment is the need to precisely specify locations *relative to the boundaries* of blocks within the possible dynamic programming matrices between feasible strings for A and B. In essence the block submatrices are rigid rectangles whose intervening gap pillars and gap beams can expand or contract. We will refer to the block submatrix for contig $A.c_i$ versus $B.c_j$ simply as block (i, j). Position p on a left or right boundary will be the cell p symbols from the top border. If p is negative then it refers to a cell in the gap beam above the block along the column containing the relevant border, $-p$ cells above the top border. Position p on a top or bottom boundary will be the cell p symbols from the left border. If p is negative then it refers to a cell in the gap pillar left of the block, $-p$ cells left of the left border.

We start with the basic subproblem of connecting two blocks across a single gap pillar. Consider position p on the right border of block (a, b). Assuming one can adjust gap a and gaps b through $d - 1$ by Δ standard deviations, what positions on the left boundary of block $(a + 1, d)$ can you reach? The highest position is reached by making gap a as small as possible and all the gaps between $B.c_b$ and $B.c_d$ as large as possible. Following the diagonal from point p, this takes one to position $Low_{(a,b,p)}(d) = (p + (A.\mu_a \ominus \Delta A.\sigma_a)) - P_b^d$ on the left boundary of block $(a+1, d)$ where $P_b^d = \Sigma_{j=b}^{d-1}(|B.c_j| + B.\mu_j + B.\sigma_j)$. Similarly the lowest position would be $Hgh_{(a,b,p)}(d) = (p + A.\mu_a + \Delta A.\sigma_a) - M_b^d$ where $M_b^d = \Sigma_{j=b}^{d-1}(|B.c_j| + (B.\mu_j \ominus B.\sigma_j))$. Of course, one can under- and over-shoot the left boundary of block $(a + 1, d)$. So the set of positions reached are those in $[Low_{(a,b,p)}(d), Hgh_{(a,b,p)}(d)] \cap [0, |B.c_d|]$.

Symmetric equations and conditions apply for the subproblem of connecting blocks across a single gap beam. Moreover, note that $Low_{(a,b,p)}(d+1) \leq Low_{(a,b,p)}(d) - |B.c_d|$ and $Hgh(a, b, p)(d+1) \leq Hgh_{(a,b,p)}(d) - |B.c_d|$. Thus both these functions are monotone decreasing in d and if one of the values is in the interval $[0, |B.c_d|)$ for a given value of d, then the $d + 1$'st value is less than 0. Therefore, the range of d for which a point on the left boundary of block $(a + 1, d)$ is reached form an interval $[e, f]$ where if $e < f$ the points reached are a suffix of the boundary of block $(a+1, e)$, all of the boundary of blocks $(a + 1, e + 1)$ through $(a + 1, f - 1)$ (if any), and a prefix of boundary block $(a + 1, f)$.

> *Lemma:* If gaps are restricted to be non-negative then the blocks reachable across a gap pillar from a point on a given block constitute an interval and can be found in $O(\log n_B)$ time using binary search.

The next step is to consider connections that traverse not only gap pillar a, but enter gap beam d to connect to the top of block $(a+1, d+1)$ and beyond. To do so it suffice to determine what positions left of the bottom boundary of

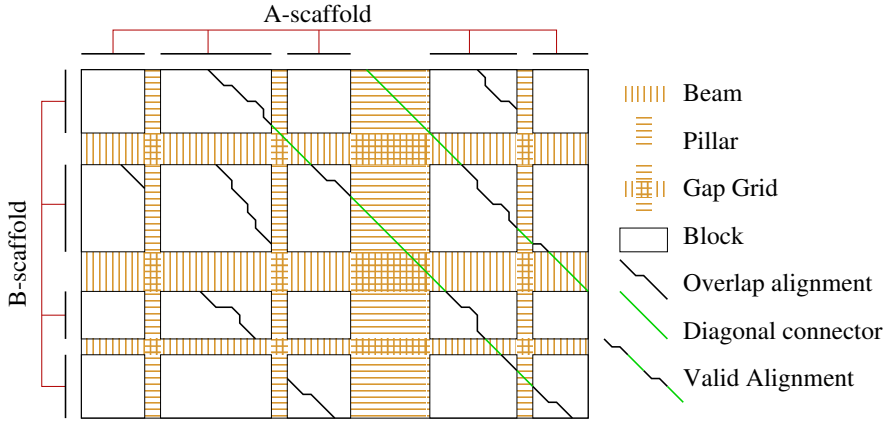

Figure 2: Blocks, Pillars, Beams, Gap Grid, and Valid Alignments.

block $(a+1,d)$ are reachable from position p on the right boundary of (a,b). We call this interval of gap entry positions, *beam portal* (a,d). Recall that these positions are negative and are the cells on the border of gap beam d at which a connection diagonal can enter the beam. Thereafter one can recursively solve the reachability problem for the beam, reaching the tops of blocks and entering gap pillars from which one can again recurse. The leftmost position is reached by making gap pillar a as large as possible and gap beams b through $d-1$ as small as possible. Following the diagonal from point p, takes one to the position $|B.c_d| - Hgh_{(a,b,p)}(d)$ on the bottom border of block (a,d). Similarly it follows that $|B.c_d| - Low_{(a,b,p)}(d)$ is the rightmost position reachable on this border. While one can reach points further left than $-slim_d = -(B.\mu_d \ominus \Delta B.\sigma_d)$ there is no need to consider them as one can reach position 0 on the top of block $(a+1,d+1)$ from this point. We thus consider the interval $[-slim_d, 0]$ as the set of possible points constituting beam portal (a,d). However, one must be careful because in some cases one reaches only points in the interval $[-wide_d, -slim_d)$ where $wide_d = B.\mu_d + \Delta B.\sigma_d$, in which case one must and can equivalently assert that the point $-slim_d$ has been reached. To this end we define

$$Low^{\star}_{(a,b,p)}(d) = \begin{cases} |B.c_d| + slim_d & \text{if } Low_{(a,b,p)}(d) \in \\ & \quad [|B.c_d| + slim_d, \\ & \quad\ |B.c_d| + wide_d] \\ Low_{(a,b,p)}(d) & \text{otherwise} \end{cases}$$

One should observe that $Low^{\star}$ is still monotone decreasing and always not greater than the corresponding Hgh value. So accounting for over- and under-shooting the relevant range, the set of positions at which one can reach beam portal d through some instantiation of gap pillar a, are those in $[|B.c_d| - Hgh_{(a,b,p)}(d), |B.c_d| - Low^{\star}_{(a,b,p)}(d)] \cap [-slim_d, 0]$. We leave it as an exercise to verify the following lemma:

> *Lemma:* If gaps are restricted to be non-negative then the beam portals reachable across a gap pillar from a point on a given block constitute an interval and can be found in $O(\log n_B)$ time.

One should now observe that the general subproblem of crossing a gap pillar or beam has as input not a single position but an interval of positions along a right or bottom boundary. Given such an interval $[p,q]$ the intervals of positions reached on a diagonal trajectory are simply $[Low_{(a,b,p)}(d), Hgh_{(a,b,q)}(d)]$ in the case of reaching blocks, and $[|B.c_d| - Hgh_{(a,b,q)}(d), |B.c_d| - Low^{\star}_{(a,b,p)}(d)]$ in the case of entering gap beams or pillars. We put the pieces together in the algorithm sketch of Figure 3 and illustrate the operation of this reaching computation with a computer generated depiction of its operation on a problem instance in Figure 4. In particular, one should note that we treat the problem of computing edges from Θ and to Φ by starting searches from the top of each pillar and the left end of each beam, and when searching, noting that one has reached the bottom of a pillar or the right end of a beam.

It remains to explore the issue of efficiency. To start, observe that if one precomputes n_B-element arrays, $Psum[i] = \Sigma_{j=1}^{i-1}(|B.c_j| + B.\mu_j + \Delta B.\sigma_j)$ and $Msum[i] = \Sigma_{j=1}^{i-1}(|B.c_j| + (B.\mu_j \ominus \Delta B.\sigma_j))$, in $O(n_B)$ time, then one can compute $Low_{(a,b,p)}(d) = (p + A.\mu_a \ominus \Delta A.\sigma_a) - (Psum[d] - Psum[b])$ and $Hgh(a,b,p)(d) = (p + A.\mu_a + \Delta A.\sigma_a) - (Msum[d] - Msum[b])$ in constant time on demand.

Presented as a recursive search routine above, the time for the algorithm is equal to the number of recursive calls and reached blocks times a logarithmic factor. In the practical situations we see, gaps are generally quite narrow compared to blocks and most searches involve a constant number of blocks and recursive calls. However, this recursion from a worst case point of view could potentially involve exponential time for ill-posed cases. So we conclude this section with a restructuring of the reaching steps to give an algorithm with an $O(n_A n_B (n_A + n_B)\log(n_A + n_B))$ worst-case time to find an edge, primarily for its theoretical value in demonstrating that the problem is polynomial.

First observe that there are $(n_A - 1)(n_B - 1)$ beam and pillar portals. The (a,b) beam portal is reachable from either a pillar portal for a block $(a-1,d)$ for $d \le b$ or the vertex γ if its overlap ends on the right boundary of such a block. Similarly what portion of the (a,b) pillar portal is covered depends on what portion of beam portals $(c,b-1)$ for $c \le a$ are covered and if vertex γ's overlap ends on the bottom boundary of such a block. The graph of these dependencies is acyclic implying that one can order the computation so the one does not call the $Traverse$ routine on a portal until all its predecessors have been processed and thus when all possible entry points into the portal are knowable. In fact, processing portals in row or column major order of their affiliated blocks suffices.

We need to conceptually simplify the notion of the regions reached across a pillar or beam from a given portal interval or block position. To do so we move from the rela-

```
global γ: local alignment
       G: stretch graph

Procedure Traverse_Pillar((a, b): block, [p, q]: interval)
  { if [p, q] = ∅ or a ≥ n_A then
      { if [p, q] ≠ ∅ then "Make an edge from γ to Φ in G"
        return
      }
    for d ← b ... n_B do
      { x ← Low_(a,b,p)(d)
        y ← Hgh_(a,b,q)(d)
        for τ s.t. τ.a = a + 1 and τ.b = d and −τ.beg ∈ [x, y] ∩ [0, |B.c_d|] do
            "Make an edge from γ to τ in G"
        x ← Low*_(a,b,p)(d)
        Traverse_Beam((a + 1, d),[|B.c_d| − y, |B.c_d| − x] ∩ [−(B.μ_d ⊖ ΔB.σ_d), 0])
      }
  }

Procedure Traverse_Beam((a, b): block, [p, q]: interval)
  { "Symmetric to Traverse_Pillar" }

"Initialize stretch graph G with no edges"
γ ← Θ
for j ← 1 ... n_B − 1 do
    Traverse_Pillar((1, j),[−(B.μ_j ⊖ ΔB.σ_j), 0])
for i ← 1 ... n_A − 1 do
    Traverse_Beam((i, 1),[−(A.μ_i ⊖ ΔA.σ_i), 0])
for γ ∈ ϒ do
    if γ.end ≥ 0 then
        Traverse_Pillar((γ.a, γ.b),[|B.c_γ.b| − γ.end, |B.c_γ.b| − γ.end])
    else
        Traverse_Beam((γ.a, γ.b),[|A.c_γ.a| + γ.end, |A.c_γ.a| + γ.end])
```

Figure 3: The Basic Edge Determination Algorithm.

tive positioning scheme used above to an absolute one. Map point p on the left boundary of block (a, d) to absolute position $M_1^d + p$ and map point p in beam portal (a, d) to $M_1^d + |B.c_d| - p$. Every unique point on left block boundaries and beam portals involving contig a of A are mapped isomorphically to the domain $[0, M_1^{n_B - 1} + |B.c_{n_B}|]$. Moreover, the result of projecting an interval of points across pillar $a - 1$ is an interval of this range in the absolute system by our previous lemmas. Thus we may now think of the result of propagating an interval across a beam or pillar as resutling in an interval of reached points.

To obtain an efficient ordering of the reached boundaries, we associate an interval tree [10] with every row and column of the blocks. When pushing from an interval across a beam or column, we compute the absolute-coordinate interval reached and add it to the appropriate interval tree for the boundary and portal positions on the far side of the gap in $O(\log(n_A + n_B))$ time. When it is time to process a portal or block boundary, the set of intervals covering the object's range are extracted in $O(\log(n_A + n_B))$ time per interval from the relevant interval tree. It only remains to argue that at most $O(n_A n_B(n_A + n_B))$ intervals are extracted from interval trees during the course of execution.

The regions of a given portal or block boundary that are reached from γ constitute a collection of disjoint intervals and we distinguish three types: (a) *whole intervals* that cover the entire object's range, (b) *end intervals* that cover a suffix or prefix of the object's boundary, and (c) *interior intervals* that are strictly interior to the object's boundary. Clearly there can be at most $O(n_A n_B)$ whole and end intervals, as there can be at most two such intervals per portal and block boundary. Next observe that if propagating portal interval u produces an interior interval at a portal across its beam or pillar, then u cannot produce intervals at any of its other successors in the portal dependency graph. Thus every interior interval "consumes" its successors. A chain of dependent interior intervals can pass through at most $O(n_A + n_B)$ portals and must initially start at a whole or end interval of which there are only $O(n_A n_B)$ to consume. Thus there are at most $O(n_A n_B(n_A + n_B))$ interior intervals.

We thus have an $O(\Upsilon n_A n_B(n_A + n_B)\log(n_A + n_B))$ algorithm for building the stretch graph for a sparse scaffold comparison problem. It remains an open problem to design a better worst-case algorithm. For the simple reaching-based algorithm of Figure 3 we will show in empirical trials that performance is $O(\Upsilon)$ for our real-world applications. This leaves open the problem of characterizing expected-case performance as a function of some parameterization of problem instances.

4 The Stretch Optimization Algorithm

In the previous section we gave a practical algorithm for finding all edges in the stretch graph and further demonstrated that the problem was polynomial. Given the stretch graph one can then in $O(|\Upsilon|)$ time compute an optimal path from Θ to Φ where each vertex is weighted by the δ-cost of its overlap alignment. Moreover, it is well-known [11] how to compute the set of edges that are on an optimal path in the same time. Every path over the set of edges in this subgraph is an optimal path, so we can solve the optimal stretch problem if we can determine the minimum weight path through this subgraph when each edge is weighted according to its *minimum stretch* weight. This follows as the set of gaps involved in the connection of a given edge are disjoint from all others on any Θ to Φ path. So we need only solve the problem of finding the minimum stretch weight of each edge in the subgraph of optimum paths.

Consider edge $\gamma \rightarrow \tau$. The algorithms of the previous section determined all the portal and block boundary positions reachable along a diagonal connector from γ's overlap end position in some feasible d.p. matrix. By running the same algorithm in the reverse direction one can determine all positions reachable from the start position of τ's overlap. Taking the intersection of these two sets of positions, one obtains the set of positions Π that are on a diagonal con-

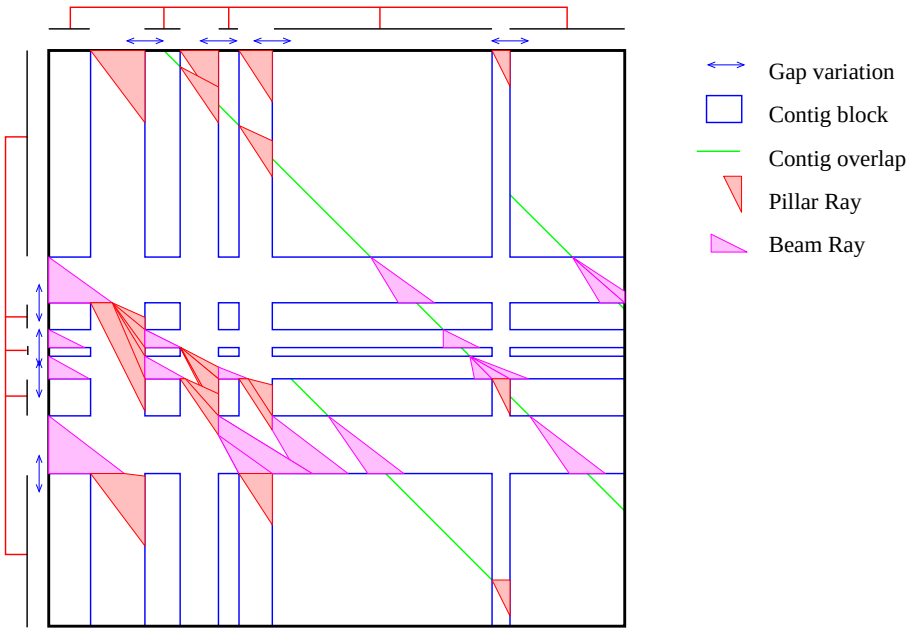

Figure 4: Depiction of Edge Determination Algorithm.

nector from γ to τ in some feasible d.p. matrix. It is clear we need only restrict our attention to the positions in Π.

A connector between γ and τ starts at γ's end point, passes through an alternating chain of beam and pillar portals, and then ends at τ's start point. We can thus decompose the problem into determining how to minimally stretch the gaps involved in each hop of this chain. Suppose one moves from position p relative to block (a, b) across gap pillar a to position q relative to block $(a+1, d)$. The total deformation required in the gap pillar a and gap beams b through $d-1$, is given by:

$$Deform_{(a,b\rightarrow d)}(p,q) = (p+A.\mu_a)-(q+\Sigma_{j=b}^{d-1}(|B.c_j|+B.\mu_j))$$

The sum of the squares of the deformation on each of the $k = (d-b)+1$ springs involved is minimized by deforming each spring equally. However, recall that each spring may be stretched at most Δ standard deviations from its resting state. So in general, the optimum is achieved by maximally stretching the springs with the smallest limitations and equally deforming all other strings equally. Suppose that the limit on stretch of the k springs is $w_1, w_2, ...w_k$ where they have been sorted so that $w_i \geq w_{i+1}$ for all i. Then the minimal stretch energy to achieve a total deformation of x units among the k strings is given by:

$$\phi_{[1,k]}(x) = \begin{cases} (x/k)^2 & \text{if } x/k \leq \omega_k \\ \omega_k^2 + \phi_{[1,k-1]}(x-\omega_k) & \text{otherwise} \end{cases}$$

One should note that you cannot, in general, simply consider the springs between γ and τ and minimally distributed stretch across them as the diagonal connector in the selected feasibility d.p. matrix may not lie in the gap grid.

To solve for the minimum stretch weight of edge $\gamma \rightarrow \tau$ we perform a dynamic programming computation over all the positions in Π. For each position $q \in \Pi$ we compute, $S[q]$, the minimum stretch weight of the springs from the end of γ's overlap to the position q through all possible connection paths over Π. We do so in row- or column-major order of the blocks associated with the positions. For position q, $S[q]$ is the minimum of $S[p] + \phi(Deform(p,q))$ over all direct predecessor positions p that are across a gap from q. One can further optimize the computation of ϕ so that it takes $O(1)$ time to incrementally compute for successive value of p.

It takes $O((A.\mu_i + \Delta A.\sigma_i)(B.\mu_j + \Delta B.\mu_j))$ worst case time to compute the S-values for positions in a portal for pillar i or beam j reachable from those in a portal for beam j or pillar i. In the worst case, all such pairings occur in evaluating an edge, for a total worst case complexity of $O(VW)$ time where $V = \Sigma_i A.\mu_i + \Delta A.\sigma_i$ and $W = \Sigma_j B.\mu_j + \Delta B.\sigma_j$, are the sums of the lengths of the maximal A and B gaps, respectively. On the other hand, one should observe that in practice only a linear number of optimal edges arise, and the connection hops between portals for each edge are also linear. Further restricting the positions considered to those in Π limits the quadricity of the problem even further.

5 Empirical Results

While we focused on the problem in which all gaps were proper, we found that in practice permitting negative gaps was desirable. Such a gap, if sufficiently large, allows two small contigs in a scaffold to reverse their positions. Moreover, small negative gaps model situations in which two consecutive contigs have a small overlap that might be missed or would be considered insignificant by our overlap detection software. So we permitted the use of a negative gap provided that if it implies an overlap between contigs, then the overlap must be observable between the sequences of the contigs. Incorporating this into the basic algorithm as described in Figure 3 only requires verification of contig overlaps, the algorithm otherwise handles this variation as presented. What becomes an open problem, that we do not address here, is whether the edge detection problem remains polynomial.

To understand the empirical behavior of the basic reaching algorithm of Figure 3, we built a simple simulator that

Edges	*Contigs*		
Gaps	5	10	20
10%	1.00	1.00	1.02
20%	1.03	1.10	1.11
30%	1.58	1.23	1.27
40%	2.32	1.69	1.85
50%	2.36	3.18	4.07

(a)

Blocks	*Contigs*		
Gaps	5	10	20
10%	2.44	2.70	2.88
20%	3.26	4.06	4.39
30%	9.32	6.58	7.16
40%	17.31	15.03	17.51
50%	16.74	41.93	87.65
Max.	80	360	1440

(b)

Portals	*Contigs*		
Gaps	5	10	20
10%	.41	.39	.39
20%	.87	1.14	1.22
30%	4.48	2.59	2.86
40%	9.27	7.77	8.95
50%	.9.07	24.30	51.83
Max.	40	180	720

(c)

Table 1: Empirical results.

builds two overlapping scaffolds. One can specify the span of the scaffolds, the length of the overlap (of the underlying sequences), the number of contigs per scaffold, the percent of the scaffold that is in gaps versus in contigs, the total variation in gap lengths, and the error rate of the sequence in the contigs. For the trials leading to Tables 1.a-c we generated scaffolds that span 100Kbp, overlap by 80Kbp, have a 2% sequencing error rate, and for which the gaps can vary in total by one-half of the total average length of the gaps. In the experiments we varied the number of contigs per scaffolds (5, 10, and 20, respectively) and we varied the percent of the scaffold that was in gaps (10, 20, 30, 40, and 50%, respectively). Each data point is the average value over 100 trials.

For these datasets, the number of vertices in the stretch graph is always between 1 and $n_A + n_B + 1$, and averages $1 + .8(n_A + n_B)/2$ because of the choice of 80% overlap. We present in Table 1.a the average number of outgoing edges found from each non-Φ vertex. In Table 1.b, we present the average number of block boundaries reached from an initial call on *Traverse* from each such vertex. In Table 1.c, we present the average number of gap portals reached from an initial call on *Traverse* from each vertex. This last table reflects the number of recursive calls that are made on *Traverse*. The main conclusion is that the time per edge is basically constant for any fixed level of completeness of the scaffolds, with the constant decreasing with the degree of completeness. In our applications scaffolds were an average of only 5% gaps and so performance was excellent. In fact, compute time is basically dominated by the underlying computation of the overlap alignments between contigs. So much so that once we have the contig overlaps, we solve a series of problems for Δ from 1, 2, through 10, stopping at the minimum Δ for which a scaffold overlap is found (i.e., a Θ to Φ path exists in the stretch graph). This gives us a rough estimate of the amount of stretch required to get the scaffolds to overlap.

Acknowledgements: The author wishes to thank Laurent Mouchard and Aaron Halpern for their assistance in the practical aspects of this work and the critical evaluation of the results.

References

[1] U.S. Dept. of Energy, Office of Energy Research and Office of Biological Environmental Research. Human Genome Program Report. `http://www.ornl.gov/hgmis/publicat/97pr/` (1997).

[2] J. Weber, E. Myers *Genome Research* **7** (1997).

[3] J.C. Venter, M.D. Adams, G.G. Sutton, A.R. Kerlavage, H.O. Smith, and M. Hunkapillar. Shotgun sequencing of the human genome. *Science 280* (1998), 1540-1542.

[4] E.S. Lander, M.S. Waterman. Genomic mapping by fingerprinting random clones: a mathematical analysis. *Genomics 2* (1988), 231-239.

[5] M.R. Olson, L. Hood, C. Cantor, and D. Botstein. A common language for physical mapping of the human genome. *Science 245* (1989), 1434-1435.

[6] D. Gusfield. *Algorithms on Strings, Trees, and Sequences.* (Cambridge University Press, 1997), Chapter 11.

[7] E. Myers, W. Miller. Approximate matching of regular expressions. *Bull. of Mathematical Biology 51* (1989), 5-37.

[8] S.F. Altschul, W. Gish, W. Miller, E. Myers, D.J Lipman. A basic local alignment search tool. *J. Mol. Biol. 215* (1990), 333-340.

[9] J. O'Rourke *Art Gallery Theorems and Algorithms* (Oxford University Press, 1987).

[10] F.P. Preparata, M.I. Shamos *Computational Geometry: An Introduction* (Springer-Verlag, 1985).

[11] D. Naor, D. Brutlag. On suboptimal alignments of biological sequences. *Proc. 4th Symp. on Combinatorial Pattern Matching* (Padova, Italy 1994), 179-196.