Chapter 1

A SIMPLE APPROXIMATION ALGORITHM FOR NONOVERLAPPING LOCAL ALIGNMENTS (WEIGHTED INDEPENDENT SETS OF AXIS PARALLEL RECTANGLES)

Piotr Berman
Department of Computer Science & Engineering
Pennsylvania State University
University Park, PA 16802
berman@cse.psu.edu

Bhaskar DasGupta*
Department of Computer Science
Rutgers University
Camden, NJ 08102
bhaskar@crab.rutgers.edu

Abstract We consider the following problem motivated by applications to nonoverlapping local alignment problems in computational molecular biology: we are a given a set of n positively weighted axis parallel rectangles such that, for each axis, the projection of a rectangle on this axis does not enclose that of another, and our goal is to select a subset of *independent* rectangles from the given set of rectangles of total maximum weight, where two rectangles are independent provided for each axis, the projection of one rectangle does not overlap that of another. We use the two-phase technique of [3] to provide a simple approximation algorithm for this problem that runs in $O(n \log n)$ time with a worst-case performance ratio of 3. We also discuss extension and analysis of the algorithm in d dimensions.

*Supported in part by NSF grant CCR-9800086.

Keywords: Computational Biology, Nonoverlapping Local Alignments, Approximation Algorithms.

1. INTRODUCTION

A fundamental problem that arises in the comparison of genomic sequences in computational molecular biology for similarity or dissimilarity is to select fragments of high local similarity between two strings [6]. Motivated by this application, Bafna et al. [1], considered the following maximization problem, termed as the Independent subset of Rectangles (IR) problem. We are a given a set S of n positively weighted axis parallel rectangles such that, for each axis, the projection of a rectangle on this axis does not enclose that of another. Define two rectangles to be independent if for each axis, the projection of one rectangle does not overlap that of another. The goal of the IR problem is to select a subset $S' \subseteq S$ of *independent* rectangles from the given set of rectangles of total maximum weight. The *unweighted* version of the IR problem is the one in which the weights of all rectangles are identical. See Figure 1.1 for an pictorial illustration of the problem. The reader is referred to Section 2 of [1] for a detailed description of the relationship of this problem to the local alignment methods; Figure 1.2 shows a pictorial illustration of the relationship of a rectangle to local similarity between two fragments of two sequences.

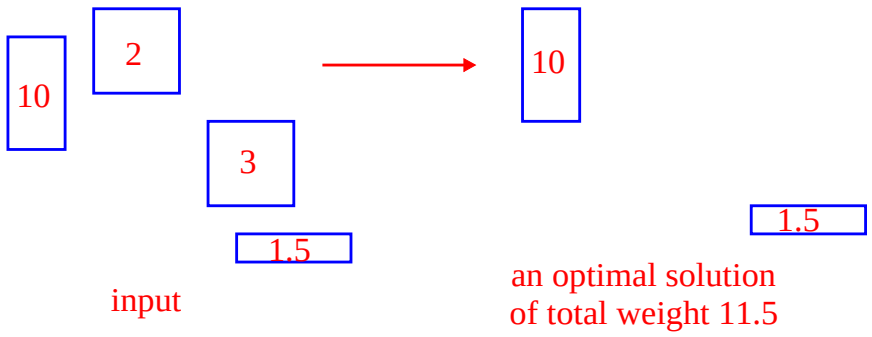

Figure 1.1. An illustration of the IR problem

A summary of previous results on this problem is as follows. Halldórsson [5] provided a polynomial time approximation algorithm with a performance ratio of $2 + \varepsilon$ (for any constant $\varepsilon > 0$) for the un-

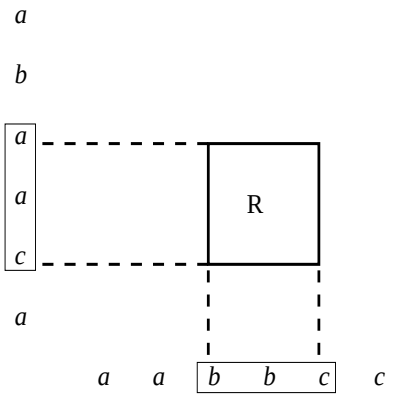

Figure 1.2. The rectangle R captures the local similarity (match) between the fragments *aac* and *bbc* of the two sequences; weight of R is the strength of the match.

weighted version of the IR problem[1]. Bafna et al. [1] showed that an approach similar to that in [5] yields a polynomial time approximation algorithm with a performance ratio of $\frac{13}{4}$ for the IR problem. The current best approximation algorithm for the IR problem is due to Berman [2] which has a performance ratio of $\frac{5}{2} + \varepsilon$ (for any constant $\varepsilon > 0$).

Consider the graph G formed from the given rectangles in which there is a node for every rectangle with its weight being the same as that of the rectangle and two nodes are connected by an edge if and only if their rectangles are *not* independent. It is not difficult to see that G is a 5-claw free graph [1] and the IR problem is tantamount to finding a *maximum-weight* independent set in G. Previous approaches have used this connection of the IR problem to the 5-claw free graphs to provide better approximation algorithms by giving improved approximation algorithms for d-claw free graphs. Most of these algorithms essentially start with an arbitrary solution and then allows small improvements to enhance the approximation quality of the solution. In contrast, we consider the IR problem directly and use the simple greedy two-phase technique of [3] to provide an approximation algorithm for the IR problem that runs in $O(n \log n)$ time with a performance ratio of 3. Although our approximation algorithm does not improve the worst-case performance ratios of previously best algorithms, it is simple to implement (involving

[1]For this and other previous approximation algorithms with an ε in the performance ratio, the running time increases with decreasing ε, thereby rendering these algorithms impractical if ε is small. Also, a straightforward implementation of these algorithms will run in at least $\Omega(n^2)$ time.

standard simple data structures such as stacks and binary trees) and runs faster than the algorithms in [1, 2, 5].

The following notations and terminologies are used for the rest of this paper. An interval $[a, b]$ is the set $[a, b] = \{x \in \mathbb{R} : \ a \leq x \leq b\}$. A rectangle R is $[a, b] \times [c, d]$ for some two intervals $[a, b]$ and $[c, d]$, where $\times$ denotes the Cartesian product. The weight of a rectangle R is denoted by $w(R)$. The *performance ratio* of an approximation algorithm for the IR problem is the ratio of the total weights of rectangles in an optimal solution to that in the solution provided by the approximation algorithm. We assume that the reader with familiar with standard techniques and data structures for the design and analysis of algorithms such as in [4].

2. APPLICATION OF THE TWO-PHASE TECHNIQUE TO THE IR PROBLEM

Let $R_1, R_2, \ldots, R_n$ be the n input rectangles in our collection, where $R_i = X_i \times Y_i$ for some two intervals $X_i = [d_i, e_i]$ and $Y_i = [f_i, g_i]$. Consider the intervals $X_1, X_2, \ldots, X_n$ formed by projecting the rectangles on one axis and call two intervals X_i and X_j independent if and only if the corresponding rectangles R_i and R_j are independent. The notation $X_i \simeq X_j$ (respectively, $X_i \not\simeq X_j$) is used to denote if two intervals X_i and X_j are independent (respectively, not independent).

To simplify implementation, we first sort the set of numbers $\{d_i, e_i \mid 1 \leq i \leq n\}$ (respectively, the set of numbers $\{f_i, g_i \mid 1 \leq i \leq n\}$) and replace each number in the set by its rank in the sorted list. This does not change any feasible solution to the given problem; however, after this $O(n \log n)$ time preprocessing we can assume that $d_i, e_i, f_i, g_i \in \{1, 2, \ldots, 2n\}$ for all i. This assumption simplifies the design of data structures for the IR problem.

Now, we adopt the two-phase technique of [3] on the intervals $X_1, X_2, \ldots, X_n$. The precise algorithm is shown below in Figure 1.3. The solution to the IR problem consists of those rectangles whose projections are returned in the solution at the end of the selection phase.

The main theorem of this section is as follows.

Theorem 1 *Algorithm TPA-IR provides a correct solution to the IR problem in $O(n \log n)$ time with a performance ratio of 3.*

Before proving the performance guarantees of Theorem 1, we will first prove the correctness and running time of Algorithm TPA-IR.

Lemma 2 *Algorithm TPA-IR returns a correct solution to the IR problem in $O(n \log n)$ time.*

```
(* definitions *)
    a triplet (α, β, γ) is an ordered sequence of
        three values α, β and γ;
    L is sequence that contains a triplet (w(R_i), d_i, e_i)
        for every R_i = X_i × Y_i with X_i = [d_i, e_i];
        L is sorted so the values of e_i's are in non-decreasing order;
    S is an initially empty stack that stores triplets;
    TOTAL(X_j) returns the sum of v's of those triplets (v, a, b) ∈S
        such that [a, b] ≱ X_j;

(* evaluation phase *)
    for ( each (w(R_i), d_i, e_i) from L )
    {
        v ← w(R_i) − TOTAL([d_i, e_i]);
        if ( v > 0 )
            push((v, d_i, e_i),S);
    }

(* selection phase *)
    while ( S is not empty )
    {
        (v, d_i, e_i) ← pop(S);
        if ( [d_i, e_i] ≃ X for every interval X in our solution )
            insert [d_i, e_i] to our solution;
    }
```

Figure 1.3. Algorithm TPA-IR: Adoption of the two-phase technique for the IR problem.

Proof. To show that the algorithm is correct we just need to show that the selected rectangles are mutually independent. This is obviously ensured by the final selection phase.

It takes $O(n \log n)$ time to create the list **L** by sorting the endpoints of the n rectangles. It is easy to see that, for each interval $X_i = [d_i, e_i]$ $(1 \leq i \leq n)$, the algorithm performs only a constant number of operations, which are elementary except for the computation of TOTAL(X_i) in the evaluation phase. We need to show how this function can be computed in $O(\log n)$ time for each X_i. Note that $X_i \not\simeq X_j \equiv (X_i \cap X_j \neq \emptyset) \vee (Y_i \cap Y_j \neq \emptyset)$. Since the intervals are considered in non-decreasing order of their endpoints, there is no interval in **S** with an endpoint to the right of e_i

when TOTAL(X_i) is computed. As a result, when the computation of TOTAL(X_i) is needed, $X_i \not\supseteq X_j$ for an X_j currently in stack provided *exactly* one of the following two conditions is satisfied:

(a) $e_j \geq d_i$,

(b) $(e_j < d_i) \wedge (Y_i \cap Y_j \neq \emptyset)$.

Since for any two intervals $Y_i = [f_i, g_i]$ and $Y_j = [f_j, g_j]$, such that neither interval encloses the other, $[f_i, g_i] \wedge [f_j, g_j] \neq \emptyset$ is equivalent to either $f_i \leq f_j \leq g_i$ or $f_i \leq g_j \leq g_i$ but not both, it follows that for the purpose of computing TOTAL(X_i) it suffices to maintain a a data structure $\mathcal{D}$ for a set of points in the plane with coordinates from the set $\{1, 2, \ldots, 2n\}$ such that the following two operations can be performed:

Insert(v, x, y)**:** Insert the point with coordinates (x, y) (with $x, y \in \{1, 2, \ldots, 2n\}$) and value v in $\mathcal{D}$. Moreover, if Insert(v, x, y) precedes Insert(v', x', y'), then $y' \geq y$.

Query(a, b, c)**:** Given a query range (a, b, c) (with $a, b, c \in \{1, 2, \ldots, 2n\} \cup \{-\infty, \infty\}$), find the sum of the values of all points (x, y) in $\mathcal{D}$ with $a \leq x \leq b$ and $y \geq c$.

For example, finding all X_j's currently in stack with $e_j \geq d_i$ is equivalent to doing Query$(-\infty, \infty, d_i)$.

For notational simplicity, assume that $n = 2^k$ for some positive integer k. We start with a skeleton rooted balanced binary tree T with $2n$ leaves (and of height $O(\log n)$) in which each node will store a number in $\{1, 2, \ldots, 2n\}$. The i^{th} leaf of T (for $1 \leq i \leq 2n$) will store the point (i, y), if such a point was inserted. With each node v of T, we also maintain the following:

- a list L_v of the points stored in the leaves of the subtree rooted at v, sorted by their second coordinate. Additionally, each entry (x, y) in the list also has an additional field $sum_{x,y}$ storing the sum of all values of all points in the list to the left of (x, y) including itself, that is, the sum of values of all points (x', y') in L_v with $y' \leq y$.
- a value s_v equal to the sum of values of all points in L_v.

Initially $L_v = \emptyset$ and $s_v = 0$ for all $v \in T$ and building the skeleton tree thus obviously takes $O(n)$ time.

To implement Insert(v, x, y), we insert the point (x, y) at the x^{th} leaf of T and update L_v and s_v for every node v on the unique path from the root to the x^{th} leaf. Since Insert(v, x, y) precedes Insert(v', x', y') implies

$y' \geq y$, we simply append (x, y) to the existing L_v for every such v. It is also trivial to update $sum_{x,y}$ (from the value of $sum_{x',y'}$ where (x', y') was the previous last entry of the list) and s_v in constant time. Since there are $O(\log n)$ nodes on the above unique path, we spend $O(\log n)$ time.

To implement Query(a, b, c), we search for a and b in T (based on the first coordinates of the points only) as in a binary search tree and let v be the lowest common ancestor of the two search paths. Then L_v contains all points (x, y) with $a \leq x \leq b$. We do a binary search on L_v in $O(\log |L_v|) = O(\log n)$ time based on the second coordinate of points to find a point (x', y') with y' being the largest possible value satisfying $y' < c$. Then, the answer to our query is the quantity $s_v - sum_{(x',y')}$. ❑

Now, we prove the performance ratio of Algorithm TPA-IR as promised in Theorem 1. Let B be a solution returned by Algorithm TPA-IR and A be any optimal solution. For a rectangle $R \in A$, let β_R denote the number of those rectangles in B that were *not* independent of R *and* were examined no earlier than R by the evaluation phase of Algorithm TPA-IR and let $\beta = \max_{R \in A} \beta_R$.

Theorem 3 *Algorithm TPA-IR has a performance ratio of* β.

Proof. Consider the set of intervals $\mathbf{S}$ in the stack at the end of the evaluation phase. Let $W(A) = \sum_{R_i \in A} w(R_i)$ and $V(\mathbf{S}) = \sum_{(v,d_i,e_i) \in \mathbf{S}} v$. It was shown in Lemma 3 of [3] that the the sum of the weights of the rectangles selected during the selection phase is at least $V(\mathbf{S})$. Hence, it suffices to show that $\beta V(\mathbf{S}) \geq W(A)$.

Consider a rectangle $R_i = X_i \times Y_i \in A$ and the time when the evaluation phase starts the processing of $X_i = [d_i, e_i]$. Let TOTAL$'([d_i, e_i])$ and TOTAL$''([d_i, e_i])$ be the values of TOTAL$([d_i, e_i])$ before and after the processing of X_i, respectively.

If $w(R_i) <$ TOTAL$'([d_i, e_i])$, then X_i is not pushed to the stack and TOTAL$''([d_i, e_i])$ = TOTAL$'([d_i, e_i])$. On the other hand, if $w(R_i) \geq$ TOTAL$'([d_i, e_i])$, then X_i is pushed to the stack with a value of $w(R_i) -$ TOTAL$'([d_i, e_i])$, as a result of which TOTAL$''([d_i, e_i])$ becomes at least TOTAL$'([d_i, e_i]) + (w(R_i) -$ TOTAL$'([d_i, e_i])) = w(R_i)$. Hence, in either case TOTAL$''([d_i, e_i]) \geq w(R_i)$.

Now, summing up over all R_i's and using the definition of β and TOTAL$''([d_i, e_i])$ gives

$$
\begin{aligned}
& W(A) \\
= & \sum_{R_i \in A} w(R_i) \\
\leq & \sum_{R_i = [d_i, e_i] \times Y_i \in A} \text{TOTAL}''([d_i, e_i]) \\
\leq & \sum_{((v,a,b) \in \mathbf{S}) \wedge (b \leq e_i) \wedge ([a,b] \not\approx [d_i, e_i])} v \qquad \text{(by definition of TOTAL}''([d_i, e_i]) \\
\leq & \ \beta \sum_{(v,a,b) \in \mathbf{S}} v \qquad \text{(by definition of } \beta) \\
= & \ \beta V(\mathbf{S})
\end{aligned}
$$

❑

The proof of Theorem 1 can now be completed by proving the following Lemma.

Lemma 4 *For the IR problem, $\beta = 3$.*

Proof. First note that $\beta = 3$ is possible; see Figure 1.4.

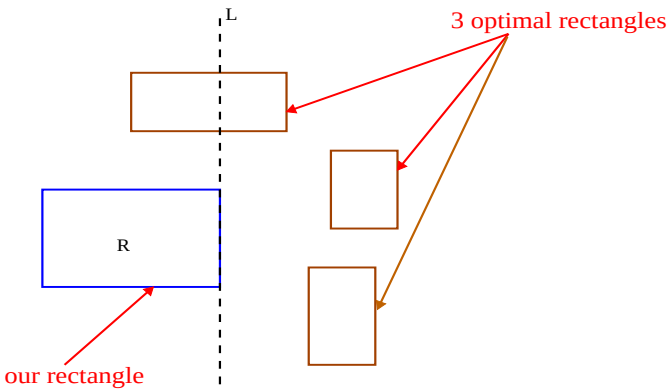

Figure 1.4. A tight example for Algorithm TPA-IR showing $\beta = 3$ is possible.

Now we show that $\beta > 3$ is impossible. Refer to Figure 1.4. Remember that rectangles in an optimal solution contributing to β must not be

independent of our rectangle R and must have their right vertical right on or to the right of the vertical line L. Since rectangles in an optimal solution must be independent of each other, there can be at most one optimal rectangle crossing L (and, thereby conflicting with R in its projections on the x-axis). Any other optimal rectangle must lie completely to the right of L and therefore may conflict with R in their projections on the y-axis only; hence there can be at most two such rectangles. ❑

3. CONCLUDING REMARKS

Algorithm TPA-IR makes a pass on the projections of the rectangles on the x-axis in a nondecreasing order of the endpoints of the projections. Can we improve the performance ratio if we run TPA-IR separately on the projections on the x-axis in left-to-right and in right-to-left order of endpoints and take the better of the two solutions? Or, even further, we may try running Algorithm TPA-IR two more times separately on the projections on the y-axis in top-to-bottom and in bottom-to-top order and take the best of the four solutions. Figure 1.5 shows that even then the worst case performance ratio will be 3. We already exploited the planar geometry induced by the rectangles for the IR problem to show that $\beta \leq 3$. Further research may be necessary to see whether we can exploit the geometry of rectangles more to design simple approximation algorithms with performance ratios better than 2.5 in the weighted case or better than 2 in the unweighted case.

In practice, the d-dimensional variation of the IR problem, motivated by the selection of fragments of high local similarity between d strings, is more important. In this version, we are given a set S of n positively weighted axis parallel d-dimensional hyper-rectangles[2] such that, for every axis, the projection of a hyper-rectangle on this axis does not enclose that of another. Defining two hyper-rectangles to be independent if for every axis, the projection of one hyper-rectangle does not overlap that of another, the goal of the d-dimensional IR problem is to select a subset $S' \subseteq S$ of *independent* hyper-rectangles from the given set of rectangles of total maximum weight. Algorithm TPA-IR can be applied in an obvious way to this extended version by considering the projections of these hyper-rectangles on a particular axis. It is not difficult to see that $\beta \leq 2^d - 1$ for this case, thus giving a worst-case performance ratio of $2^d - 1$. Whether one can design an algorithm with a performance ratio that increases less drastically (e.g., linearly) with d is still open.

[2]A d-dimensional hyper-rectangle is a Cartesian product of d intervals.

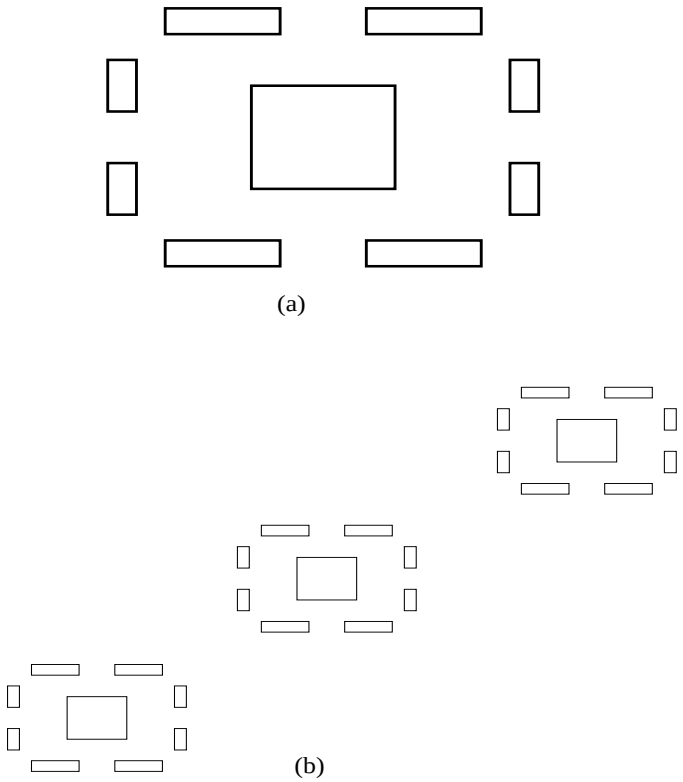

Figure 1.5. (a) The basic block of 9 rectangles such that the four runs of TPA-IR will always select R, whereas an optimal solution will select 3 of the remaining 8 rectangles. (b) The basic block repeated $\frac{n}{9}$ times such that different copies do not interfere in their projections on either axis, resulting in a performance ratio of 3 even for the best of the 4 runs.

References

[1] V. Bafna, B. Narayanan and R. Ravi, *Nonoverlapping local alignments (Weighted independent sets of axis-parallel rectangles*, Discrete Applied Mathematics, 71, pp. 41-53, 1996.

[2] P. Berman, *A $d/2$ approximation for maximum weight independent set in d-claw free graphs*, proceedings of the 7th Scandinavian Workshop on Algorithmic Theory, Lecture Notes in Computer Science, 1851, Springer-Verlag, July 2000, pp. 214-219.

[3] P. Berman and B. DasGupta, *Improvements in Throughput Maximization for Real-Time Scheduling*, proceedings of the 32nd Annual ACM Symposium on Theory of Computing, May 2000, pp. 680-687.

[4] T. H. Cormen, C. E. Leiserson and R. L. Rivest, Introduction to Algorithms, The MIT Press, 1990.

[5] M. M. Halldórsson, *Approximating discrete collections via local improvements*, proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms, January 1995, pp. 160-169.

[6] T. F. Smith and M. S. Waterman, *The identification of common molecular sequences*, Journal of Molecular Biology, 147, 1981, pp. 195-197.