2 Efficient combinatorial algorithms for DNA sequence processing

BHASKAR DASGUPTA
Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607
Email: dasgupta@cs.uic.edu

MING-YANG KAO
Department of Electrical Engineering & Computer Science
Northwestern University
Evanston, IL 60208
Email: kao@cs.northwestern.edu

2.1 INTRODUCTION

The modern era of molecular biology began with the discovery of the double helical structure of DNA. Today, sequencing nucleic acids, the determination of genetic information at the most fundamental level, is a major tool of biological research [44]. This revolution in biology has created a huge amount of data at great speed by directly reading DNA sequences. The growth rate of data volume is exponential. For instance, the volume of DNA and protein sequence data is currently doubling every 22 months [32]. One important reason for this exceptional growth rate of biological data is the medical use of such information in the design of diagnostics and therapeutics [22, 31]. For example, identification of genetic markers in DNA sequences would provide important informations regarding which portions of the DNA are significant, and would allow the researchers to find many disease genes of interest (by recognizing them from the pattern of inheritance). Naturally, the large amount of available data poses a serious challenge in storing, retrieving and analyzing biological information.

A rapidly developing area, *computational biology*, is emerging to meet the rapidly increasing computational need. It consists of many important areas such as information storage, sequence analysis, evolutionary tree construction, protein structure

prediction, and so on [22, 31]. It is playing an important role in some biological research. For example, sequence comparison is one of the most important methodological issues and most active research areas in current *biological sequence analysis*. Without the help of computers, it is almost impossible to compare two or more biological sequences (typically, at least a few hundred character long). Applications of sequence comparison methods can be traced back to the well-known *Human Genome Project* [43], whose objective is to decode this entire DNA sequence and to find the location and ordering of genetic markers along the length of the chromosome. These genetic markers can be used, for example, to trace the inheritance of chromosomes in families and thereby to find the location of disease genes. Genetic markers can be found by finding DNA polymorphisms, *i.e.*, locations where two DNA sequences "spell" differently. A key step in finding DNA polymorphisms is the calculation of the *genetic distance*, which is a measure of the correlation (or similarity) between two genomes.

In this chapter, we discuss computational complexities and approximation algorithms for a few DNA sequence analysis problems. We assume that the reader is familiar with the basic concepts of exact and approximation algorithms [20, 42], basic computational complexity classes such as P and NP [23, 26, 36] and basic notions of molecular biology such as DNA sequences [24, 45].

2.2 NONOVERLAPPING LOCAL ALIGNMENTS

As we have already seen, a fundamental problem in computational molecular biology is to elucidate similarities between sequences and a cornerstone result in this area is that given two strings of length p and q, there are local alignment algorithms that will score pairs of substrings for "similarity" according to various biologically meaningful scoring functions and we can pull out all "similar" or high scoring substring pairs in time $O(pq + n)$ where n is the output size [45]. Having found the high scoring substring pairs, a global description of the similarity between two sequences is obtained by choosing the disjoint subset of these pairs of highest total score. This problem is in general referred to as the "non-overlapping local alignment" problem. We also mention a more general "d-dimensional version" of this problem involving $d > 2$ sequences, where we score d substrings, one from each sequence, with a similarity score and the goal is to select a collection of disjoint subsets of these d-tuples of substrings maximizing the total similarity.

A natural geometric interpretation of the problem is via selecting a set of "independent" rectangles in the plane in the following manner [3]. Each output substring pair being represented as a rectangle; Figure 2.1 shows a pictorial illustration of the relationship of a rectangle to local similarity between two fragments of two sequences. This gives rise to the following combinatorial optimization problem. We are a given a set S of n positively weighted axis parallel rectangles. Define two rectangles to be independent if for each axis, the projection of one rectangle does not overlap that of another. The goal is to select a subset $S' \subseteq S$ of *independent* rectangles from the given set of rectangles of total maximum weight. The *unweighted*

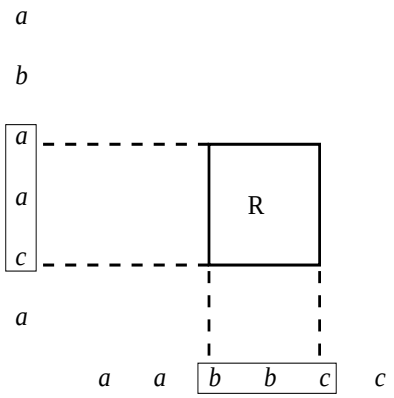

Fig. 2.1 The rectangle R captures the local similarity (match) between the fragments aac and bbc of the two sequences; weight of R is the strength of the match.

version of the problem is the one in which the weights of all rectangles are identical. In the d-dimensional version, we are given a set of positively weighted axis parallel d-dimensional hyper-rectangles[1] 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 is to select a subset of *independent* hyper-rectangles of total maximum weight from the given set of hyper-rectangles.

The non-overlapping local alignment problem, including its special case as defined by the IR problem described in Section 2.2.2, is known to be NP-complete. The best known algorithm for the general version of the nonoverlapping local alignment problem is due to [8] who provide a $2d$-approximation for the problem involving d-dimensional hyper-rectangles. In the sequel, we will discuss two important special cases of this problem that are biologically relevant.

2.2.1 The Chaining Problem

The chaining problem is the following special case [24, page 326]. A subset of rectangles is called a *chain* if no horizontal or vertical line intersects more than one rectangle in the subset in the subset and if the rectangles in the subset can be ordered such that each rectangle in this order is below and to the right of its predecessor. The goal is to find a chain of maximum total similarity. This problem can be posed as finding the longest path in a directed acyclic graph and thereby admits an optimal solution in $O(n^2)$ time where n is the number of rectangles. However, using a sparse dynamic programming method, the running time can be further improved to $O(n \log n)$ [27].

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

2.2.2 The Independent Subset of Rectangles (IR) Problem

In this problem, first formulated by [3], for each axis, the projection of a rectangle on this axis does not enclose that of another; this restriction on the input is biologically justified by a preprocessing of the input data (fragment pairs) to eliminate violations of the constraint. See Figure 2.2 for an pictorial illustration of the problem.

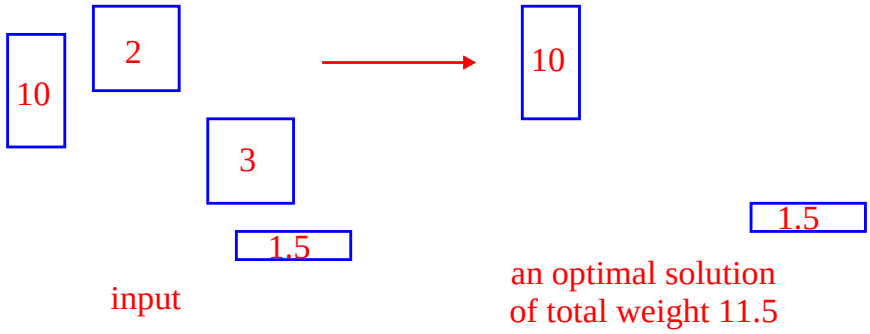

Fig. 2.2 An illustration of the IR problem

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 [3] and the IR problem is tantamount to finding a *maximum-weight* independent set in G. Many 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. For example, using this approach, Bafna et al. [3] provided a polynomial time approximation algorithm with a performance ratio[2] of $\frac{13}{4}$ for the IR problem and Halldórsson [25] provided a polynomial time approximation algorithm with a performance ratio of $2+\varepsilon$ (for any constant $\varepsilon > 0$) for the unweighted version of the IR problem[3]. The current best approximation algorithm for the IR problem is due to Berman [9] via the same approach which has a performance ratio of $\frac{5}{2}+\varepsilon$ (for any constant $\varepsilon > 0$).

Many of the abovementioned algorithms essentially start with an arbitrary solution and then allows small improvements to enhance the approximation quality of the solution. In contrast, in this section we review the usage of a simple greedy two-phase technique to provide an approximation algorithm for the IR problem with a performance ratio of 3 that runs in $O(n \log n)$ time [10, 15]. The two-phase technique

[2]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.
[3]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 $\Omega(mn)$ time.

was introduced in its more general version as a multi-phase approach in the context of real-time scheduling of jobs with deadline in [11, 12]; we review the generic nature of this technique in Section 2.2.2.1. Although this approximation algorithm does not improve the worst-case performance ratios of previously best algorithms, it is simple to implement (involving standard simple data structures such as stacks and binary trees) and runs faster than the algorithms in [3, 9, 25].

2.2.2.1 The Local-ratio and Multi-phase Techniques The multi-phase technique was introduced formally in the context of real-time scheduling of jobs by the investigators in [11, 12]. Informally and very briefly, this technique works as follows:
(a) We maintain a stack **S** containing objects that are tentatively in the solution. **S** is initially empty before the algorithm starts.
(b) We make $k \geq 1$ *evaluation passes* over the objects. In each evaluation pass:

- we inspect the objects in a specific order that is easy to compute (*e.g.*, rectangles in the plane in the order of their right vertical side),
- depending on the current content of **S**, the contents of **S** during the previous passes as well as the attributes of the current object, we compute a score for the object,
- we push the object to **S** if the score is above a certain threshold

(c) We make one *selection pass* over the objects in **S** in a specific order (typically, by popping the elements of **S**) and select a subset of the objects in **S** that satisfy the feasilibilty criteria of the optimization problem under consideration.

Closely related to the two-phase version of the multi-phase technique, but somewhat of more general nature, is the *local-ratio* technique. This technique was first developed by Bar-Yehuda and Even [7] and later extended by Berman et al. [4] and Bar-Yehuda [6]. The crux of the technique is as follows [5]. Assume that given an n-dimensional vector $\vec{p}$, our goal is to find a n-dimensional *solution* vector $\vec{x}$ that maximizes (respectively, minimizes) the inner product $\vec{p} \cdot \vec{x}$ subject to some set $\mathcal{F}$ of *feasibility constraints* on $\vec{x}$. Assume that we have decomposed the vector $\vec{p}$ to two vectors $\vec{p_1}$ and $\vec{p_2}$ with $\vec{p_1} + \vec{p_2} = \vec{p}$ such that, for some $r \geq 1$ (respectively, $r \leq 1$), we can find a solution vector $\vec{x}$ satisfying $\mathcal{F}$ which r-approximates $\vec{p_1}$ and $\vec{p_2}$, that is, which satisfies both $\vec{p_1} \cdot \vec{x} \geq r \cdot \max_{\vec{y}}\{\vec{p_1} \cdot \vec{y}\}$ and $\vec{p_2} \cdot \vec{x} \geq r \cdot \max_{\vec{y}}\{\vec{p_2} \cdot \vec{y}\}$ (respectively, $\vec{p_1} \cdot \vec{x} \leq r \cdot \min_{\vec{y}}\{\vec{p_1} \cdot \vec{y}\}$ and $\vec{p_2} \cdot \vec{x} \leq r \cdot \min_{\vec{y}}\{\vec{p_2} \cdot \vec{y}\}$). Then, $\vec{x}$ also r-approximates $\vec{p}$. This allows a given problem to be recursively broken down in subproblems from which one can recover a solution to the original problem. The local-ratio approach makes it easier to extend the results to a larger class of problems, while the multi-phase approach allows to obtain better approximation ratios in many important special cases.

The multi-phase technique was used in the context of job scheduling in [11, 12] and in the context of opportunity-cost algorithms for combinatorial auctions in [1]. We will discuss the usage the two-phase version of the multi-phase approach in the context of the IR problem [10, 15] in the next section. In some cases, it is also

possible to explain the multi-phase or the local-ratio approach using the primal-dual schema; for example, see [5].

2.2.2.2 Application of the Two-Phase Technique to the IR Problem The following notations and terminologies are used for the rest of this section. 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 $w(R)$. We assume that the reader with familiar with standard techniques and data structures for the design and analysis of algorithms such as in [20].

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 on the intervals $X_1, X_2, \ldots, X_n$. The precise algorithm is shown in Figure 2.3. The solution to the IR problem consists of those rectangles whose projections are returned in the solution at the end of the selection phase.

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. To implement this algorithm, we need to compute TOTAL(X_i) efficiently. Using the fact that the intervals are considered in non-decreasing order of their endpoints, we can reduce this to the problem of maintaining 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 $y \geq c$.

One can solve this problem in $O(n \log n)$ time and space preprocessing and $O(\log n)$ per query by using an appropriately augmented binary search tree; see [10, 15] for details. We can therefore implement the entire algorithm in $O(n \log n)$ time and space.

We now sketch the main points of the proof of the performance ratio of Algorithm TPA-IR as detailed in [10, 15]. let B be a solution returned by Algorithm TPA-

(* 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 $R_i = X_i \times Y_i$ 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 $(v, a, b) \in$**S**such that $[a, b] \not\simeq X_j$;
(* evaluation phase *)
for (each $(w(R_i), d_i, e_i)$ from **L**)
{
$v \leftarrow w(R_i) -$ TOTAL$([d_i, e_i])$;
if ($v > 0$) push$((v, d_i, e_i)$,**S**);
}
(* selection phase *)
while (**S** is not empty)
{
$(v, d_i, e_i) \leftarrow$ pop(**S**);
if ($[d_i, e_i] \simeq X$ for every interval X in our solution)
insert $[d_i, e_i]$ to our solution;
}

Fig. 2.3 Algorithm TPA-IR: Adoption of the two-phase technique for the IR problem.

IR and A be any optimal solution. For a rectangle $R \in A$, let us define the *local conflict number* β_R to be 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 $\beta = \max_{R \in A} \beta_R$. First, we show that Algorithm TPA-IR has a performance ratio of β. Next, we can show that the performance ratio of Algorithm TPA-IR is 3 by showing that for the IR problem, $\beta = 3$. First note that $\beta = 3$ is possible; see Figure 2.4. Now we show that $\beta > 3$ is impossible. Refer to Figure 2.4. 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.

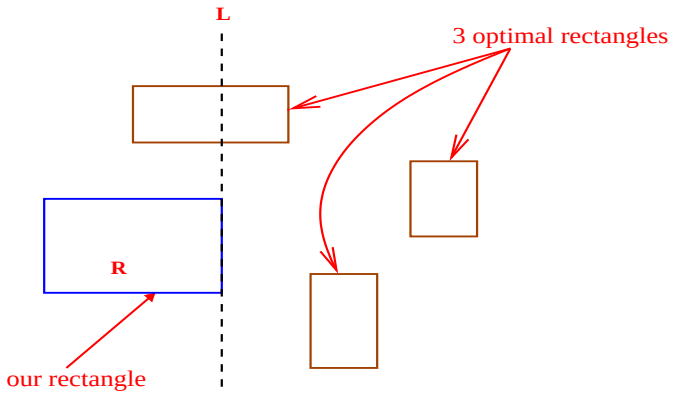

Fig. 2.4 A tight example for Algorithm TPA-IR showing $\beta = 3$ is possible.

2.2.2.3 Further Discussions 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. It is easy to draw an example that 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.

For the d-dimensional version, 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 $\beta \leq 2d - 1$ for this case [19], thus giving a worst-case performance ratio of $2d - 1$. Whether one can design an algorithm with a performance ratio that increases less drastically (*e.g.*, sublinearly) with d is still open.

2.3 GENOME TILING PROBLEMS

There are currently over 800 complete genome sequences available to the scientific community, representing the three principal domains of life: bacteria, archaea, and eukaryota [35]. Genome sequences vary widely in size and composition. In addition to the thousands of sequences that encode functional proteins and genetic regulatory elements, most eukaryotic genomes also possess a large number of *non-coding* sequences which are replicated in high numbers throughout the genome. These repetitive elements were introduced over evolutionary time and consists of families of transposable elements that can move from one chromosomal location to another, retroviral sequences integrated that are into the genome via an RNA intermediate, and simple repeat sequences that can originate de novo at any location. Nearly 50% of human genomic DNA is associated with repetitive elements. The presence of repeat sequences can be problematic for both computational and experimental biology research. For example, BLAST searches [2] with queries containing repeats against large sequence databases often result in many spurious subsequence matches, obscuring significant results and wasting computational resources. Although it is now standard practice to screen query sequences for repetitive elements, doing so subdivides the query into a number of smaller sequences that often produce a less specific match than the original. In an experimental context, when genomic sequence is used to investigate the binding of complementary DNA, repetitive elements can generate false positive signals and mask true positives by providing highly redundant DNA binding sites that compete with the meaningful targets of complementary probe sequences.

Genomic DNA can be screened for repeat sequences using specialized programs such as RepeatMasker [39] which performs local subsequence alignments [41] against a database of known repetitive elements [28]. Repeats are then masked within the query sequence, whereby a single non-nucleotide character is substituted for the nucleotides of each repeat instance. This global character replacement preserves the content and relative orientation of the remaining subsequences, which are then interspersed with character blocks representing the repetitive elements identified during the screening process.

Although the screening and/or removal of repeats is generally beneficial, additional problems may arise from the resulting genomic sequence fragmentation. Following repeat sequence identification, the remaining high-complexity component (i.e., non-repetitive DNA) exists as a population of fragments ranging in size from a few nucleotides to several kilobases. For organisms such as *Homo sapiens*, where

the genome contains many thousands of repeat elements, the vast majority of these high-complexity sequence fragments are below 1 Kb in size. This situation presents a significant impediment to both computational and experimental research. Bioinformatics analyses often benefit from the availability of larger contiguous sequences, typically 1 Kb and larger, for homology searches and gene predictions. Similarly, very small sequences (< 200 bp) are of limited use in many high-throughput experimental applications. These constraints provide the basis of the tiling problems formalized in this section.

DNA microarray design A principal motivation for looking at the tiling problems considered in this paper is their application to the design of DNA microarrays for efficient genome analysis. The microarrays we consider here are constructed from amplified genomic DNA. Each element consists of a relatively long (typically 300 bp - 1.2 Kb) sequence of genomic DNA that is acquired via the *polymerase chain reaction* (PCR) [33] in which a segment of DNA may be selectively amplified using a chemical system that recreates DNA replication *in vitro*. Although the size resolution of these array elements is not as fine as that of high-density oligonucleotide systems, PCR-based (or *amplicon*) microarrays provide experimental access to much larger regions of contiguous genomic DNA. The tiling algorithm described here has recently been used to design a microarray of this type to represent the complete sequence of human chromosome 22 [37]. When considering PCR-based microarrays, we are concerned with finding the maximum number of high-complexity subsequence fragments given a genomic DNA sequence whose repetitive elements have been identified and masked. A maximal-coverage amplicon array can then be designed by deriving an optimal tile path through the target genomic sequence such that the best set of fragments is selected for PCR amplification. Determining this tile set allows one to achieve optimal coverage of high-complexity DNA across the target sequence, while simultaneously maximizing the number of potential subsequences of sufficient size to facilitate large-scale biological research.

2.3.1 Problem Statements

Based on the applications discussed in Section 2.3, we now formalize a family of tiling problems. The following notations are used uniformly throughout the rest of the paper:

- $[i, j)$ denotes the set of integers $\{i, i + 1, \ldots, j - 1\}$;
- $[i, j] = [i, j + 1)$;
- $f[i, j)$ and $f[i, j]$ denote the elements of an array f with indices in $[i, j)$ and $[i, j]$, respectively.

Our tiling problems build upon a basic genome tiling algorithm which we call the *GTile problem* and describe as follows. The input consists of an array $c[0, n)$ of real numbers and two integer size parameters ℓ and u. A subarray $B = c[i, j)$ is called a

block of *length* $j - i$ and *weight* $w(B) = \sum_{k=i}^{j-1} c_k$, the weight of a set of blocks is the sum of their weights and a block is called a *tile* if its length belongs to $[\ell, u]$. Our goal is to find a set of pairwise disjoint tiles with the maximum possible weight. The tiling problems of interest in this paper are variations, restrictions and generalizations of the GTile problem specified by a certain combinations of the following items:

Compressed versus uncompressed input data: This is motivated by a simple binary classification of the high-complexity regions of the genome sequence from their low-complexity counterparts. Now all entries of $c[0, n)$ is either x or $-x$ for some *fixed* $x > 0$. Hence, the input sequence can be more efficiently represented by simply specifying beginnings and endings of *blocks of identical values*[4]. In other words, we can compress the input sequence $c[0, n)$ to a sequence of integers (indices) $S[0, m + 1)$ such that

- $S_0 = 0$, $S_m = n + 1$, $S_1 \geq S_0$ and $S_i > S_{i-1}$ for all $i \in [2, m]$;
- each element of $c[S_{2j}, S_{2j+1})$ is x for all $0 \leq j \leq \lfloor \frac{m}{2} \rfloor$;
- each element of $c[S_{2j-1}, S_{2j})$ is $-x$ for all $0 < j \leq \lfloor \frac{m+1}{2} \rfloor$.

We note that the input size $m + 1$ of such a compressed input data is typically *significantly smaller* than n. As a result, we can get significantly faster algorithms if we can design an algorithm for compressed inputs with a running time nearly linear in m. Furthermore, this also allows one to develop efficient hybrid approach to solving the tiling problems: first use a crude binary classification of the regions to quickly obtain an initial set of tiles and then refine the tiles taking into consideration the relative importances of the high-complexity elements.

Unbounded versus bounded number of tiles: Another important item of interest is when the number of tiles that may be used is at most a given value t, which could be considerably smaller than the number of tiles used by a tiling with no restrictions on the number of tiles. This is motivated by the practical consideration that the capacity of a microarray as obtainable by current technology is bounded.

Overlapping versus non-overlapping tiles: To enhance searching sequence databases for homology searches to allow for the case when potential matches can be found at tile boundaries, it may be useful to relax the condition of disjointness of tiles by allowing two tiles to share at most p elements for some given (usually small) $p > 0$. However, to ensure that we do not have too many overlaps, we need to *penalize* them by subtracting the weight of each overlapped region from the sum of weights of all tiles, where the *weight* of each overlapped region is the sum of the elements in it. In other words, if $\mathcal{T}$ is

[4]Notice that a $\{0, 1\}$ classification of the high-complexity regions from the low-complexity ones is not suitable since then we do not penalize for covering low-complexity regions and solving the tiling problem becomes trivial.

the set of tiles and $\mathcal{R}$ is the set of elements of $\mathbf{C}$ that belong to more than one tile in $\mathcal{T}$, then the weight is $\sum_{T\in\mathcal{T}} w(T) - \sum_{c_i\in\mathcal{R}} c_i$.

One dimensional versus d-dimensional: Generalization of the GTile problem in d dimensions has applications in database designs and related problems [16, 18, 29, 30, 34][5]. In this case, we are given a d-dimensional array $\mathbf{C}$ of size $n_1 \times n_2 \times \cdot \times n_d$ with $2d$ size parameters $\ell_1, \ell_2, \ldots, \ell_d, u_1, u_2, \ldots, u_d$, a tile is a rectangular subarray of $\mathbf{C}$ of size $p_1 \times p_2 \times \cdots \times p_d$ satisfying $\ell_i \leq p_i \leq u_i$ for all i, the weight of a tile is the sum of all the elements in the tile and our goal is again to find a set of tiles such that the sum of weights of the tiles is maximized.

We examine only those combinations of the above four items which are of importance in our applications. simplify exposition, unless otherwise stated explicitly, the GTile problem we consider is 1-*dimensional* with *uncompressed* inputs, *unbounded* number of tiles and *no overlaps*. In addition to the previously defined notations, unless otherwise stated, we use the following notations and variables with their designated meanings throughout the rest of the paper: $n + 1$ is the number of elements of the (uncompressed) 1-dimensional input array $c[i, j)$, $n_1 \leq n_2 \leq \cdots \leq n_d$ are the sizes of the dimensions for the d-dimensional input array, $w(\mathcal{T})$ is the weight for a *set* of tiles $\mathcal{T}$, t is the given number of tiles when the number of tiles is bounded and p is the maximum overlap between two tiles in 1-dimension. Finally, all logarithms are in base 2 unless stated otherwise explicitly.

2.3.2 Related Work

Tiling an array of numbers in one or more dimensions under various constraints is a very active research area (for example, see [16–18, 29, 30, 34, 40]) and has applications in several areas including database decision support, two-dimensional histogram computation and resource scheduling. Several techniques, such as the slice-and-dice approach [16], the shifting technique [26, Chapter 9] and dynamic programming methods based on binary space partitions [17, 29, 34] have proven useful for these problems. Our problems are different from the tiling problems in [16, 17, 29, 30, 34, 40]; in particular, we do not require partitioning of the entire array, the array entries may be negative and there are lower and upper bounds on the size of a tile. Other papers which most closely relate to our work are the references [38] and [46]. The authors in [38] provide an $O(n)$ time algorithm to find all *maximal* scoring subsequences of a sequence of length n. In [46] the authors investigate computing maximal scoring subsequences which contain no subsequences with weights below a particular threshold.

[5]For example, in two dimensions with $\ell_1 = \ell_2 = 0$ and $u_1 = u_2 = \infty$ this is precisely the ARRAY-RPACK problem discussed in [29].

2.3.3 Synopsis of Results

Our main theoretical results are summarized in Table 2.1; for more details, see our publications [13, 14]. All of our methods use simple data structures such as a double-ended queues and are therefore easy to implement. The techniques used for many of these tiling problems in one dimension use a solution of an *Online Interval Maximum* (OLIM) problem via a windowing scheme reminiscent of that in [21]. However, the primary consideration in the applications in [21] was reduction of space because of the online nature of their problems, whereas we are more concerned with time-complexity issues since our tiling problems are off-line in nature (and hence space for storing the entire input is always used). Moreover, our windowing scheme is somewhat different from that in [21] since we need to maintain multiple windows of different sizes and data may not arrive at evenly spaced time intervals.

Version of GTile	Time $O()$	Space $O()$	Approximation Ratio
basic	n	n	exact
overlap is from a s-subset of $[0, \delta]$, $\delta < \frac{\ell}{2}$	sn	n	exact
compressed input	$m\frac{\ell}{u-\ell}$	$m\frac{\ell}{u-\ell}$	exact
number of tiles given	$\min\{n \log \frac{n}{\ell}, nt\}$	n	exact
d-dimensional	$\left(\left(\frac{u}{\ell}\right)\varepsilon\right)^{4\left(\frac{u}{\ell}\right)^2\varepsilon^2} M\varepsilon^2$	M	$\left(1-\frac{1}{\varepsilon}\right)^d$
d-dimensional, number of tiles given	$tM + dM \log^{\varepsilon} M + dN\frac{\log N}{log \log N}$	M	$\left(\Pi_{i=1}^{d-1}\left(\lfloor 1+\log n_i \rfloor\right)\right)^{-1}$
	$M^{(2^{\varepsilon}-1)^{d-1}+1}\, dt$	$M^{(2^{\varepsilon}-1)^{d-1}+1}\, dt$	$\left(\Pi_{i=1}^{d-1}\left(\lfloor 1+\frac{\log n_i}{\varepsilon} \rfloor\right)\right)^{-1}$

Table 2.1 [13, 14] A summary of the results for the genome tiling problems. All the algorithms are either new or direct improvements of any previously known. The parameter $\varepsilon > 1$ is any arbitrary *constant*. A s-subset is a subset of s elements. For the d-dimensional case, $M = \Pi_{i=1}^{d} n_i(u_i - \ell_i + 1)$, $N = \max_{1 \le i \le d} n_i$ and $\frac{u}{\ell} = \max_i \frac{u_i}{\ell_i}$. For our biology applications $p \le 100 < \frac{\ell}{2} \ll n$, $t \simeq \frac{n}{u+\ell}$, $m \ll n$ and $\frac{\ell}{u-\ell} < 6$. The column labeled "Approximation Ratio" indicates whether the algorithm computes the optimal solution exactly or, for an approximation algorithm, the ratio of the total weight of our tiling to that of the optimum.

We also summarize the application of the GTile problem to the genomic sequences of 5 model eukaryotes. The single largest chromosome from each organism was considered as representative of the characteristics of that particular genome. Table 2.2

lists the target chromosomes and their sequence properties. The chromosomes vary in the degree of repeat density, where the first few examples contain relatively few repetitive elements in comparison to the two mammalian sequences. In the cases of *C. elegans*, *A. thaliana*, and *D. melanogaster*, the low repeat content allows us to tile the sequences fairly well simply by subdividing the remaining high-complexity DNA into sequence fragments within the appropriate size range. However, as the repeat density increases in the genomes of higher eukaryotes, so does the fragmentation of the high-complexity sequence containing genes and regulatory elements of biological significance. It soon becomes impossible to achieve maximal coverage of the high-complexity sequence in the absence of further processing.

The results of applying the tiling algorithm to each chromosome appear in Table 2.3. GTile improves the sequence coverage in all cases, easily covering nearly 100% of the high-complexity DNA in the smaller, less complex genomes with few incorporated repeats. In practice, the coverage will never reach 100% because there remains a population of small high-complexity sequences whose sizes fall below the lower bound. In terms of experimental applications, these sequences are too small to be chemically amplified by PCR and are therefore excluded from consideration.

Organism	Chromosome	Nucleotides	Repeat elements	Repetitive DNA (bp)	% Repeats (bp)	High-complexity DNA
Caenorhabditis elegans (nematode)	V	20,916,335	16,575	2,414,183	11.5	18,502,152
Arabidopsis thaliana (flowering plant)	I	30,074,119	14,490	3,557,144	11.8	26,516,975
Drosophila melanogaster (fruit fly)	3	51,243,003	27,259	3,106,633	6	48,136,370
Mus musculus (laboratory mouse)	1	196,842,934	288,551	90,532,869	46	106,310,065
Homo sapiens (human)	1	246,874,334	308,257	132,580,913	53.7	114,293,421

Table 2.2 [13, 14] Summary of target chromosome sequences. The sequences increase in repeat density with the complexity of the genome, causing a greater degree of fragmentation and loss of high-complexity sequence coverage in the unprocessed chromosomes. This situation is especially problematic in the higher eukaryotes such as human and mouse.

Acknowledgments

Bhaskar DasGupta was supported in part by NSF grants IIS-0346973, IIS-0612044 and DBI-0543365. Ming-Yang Kao was supported in part by NSF grant EIA-0112934. The authors would also like to thank all their collaborators in these research topics. Figures 2.1, 2.2, 2.3, 2.4 and the related text is included from [10] with kind permission of Springer Science and Business Media. Table 2.1, 2.2, 2.3 and the related text is included from [13] with kind permission of Mary Ann Liebert, Inc.

Target chromosome	Number of Tiles	High-complexity DNA (bp)	% Coverage	Repetitive DNA (bp)	% Repeats
		Initial sequence coverage			
C. elegans chrV	22,842	17,852,822	96.4		
A. thaliana chrI	30,075	25,972,994	98		
D. melanogaster chr3	57,568	47,366,173	98.3		
M. musculus chr1	142,165	90,988,520	85.5		
H. sapiens chr1	151,720	97,191,872	85		
		GTile, repeat penalty 6:1			
C. elegans chrV	19,034	18,299,667	99	237,772	1.28
A. thaliana chrI	25,349	26,376,577	99	196,222	0.74
D. melanogaster chr3	46,901	48,056,034	99	453,704	0.93
M. musculus chr1	128,472	96,280,008	90.5	2,314,565	2.34
H. sapiens chr1	137,403	101,866,284	89	2,026,782	1.95
		GTile, repeat penalty 5:1			
C. elegans chrV	18,975	18,329,464	99	290,152	1.55
A. thaliana chrI	25,344	26,391,095	99.5	213,917	0.8
D. melanogaster chr3	46,878	48,061,534	99.8	465,573	0.96
M. musculus chr1	127,146	97,953,586	92	4,304,560	4.2
H. sapiens chr1	136,457	103,434,234	90.4	3,788,374	3.53
		GTile, repeat penalty 4:1			
C. elegans chrV	18,891	18,345,048	99	348,086	1.86
A. thaliana chrI	25,342	26,396,637	99.5	226,559	0.85
D. melanogaster chr3	46,867	48,062,909	99.8	471,650	0.97
M. musculus chr1	125,787	98,617,314	92.7	5,765,790	5.52
H. sapiens chr1	135,305	104,138,841	91	5,247,600	4.79

Table 2.3 [13, 14] GTile results for the 5 model eukaryotic chromosomes. Maximal coverage of the high-complexity DNA is achieved with minimal repeat nucleotide inclusion, while the number of required tiles decreases. Sets of non-overlapping tiles were computed for the size range of 300 bp – 1 Kb.

REFERENCES

1. K. Akcoglu, J. Aspnes, B. DasGupta and M.-Y. Kao. *Opportunity Cost Algorithms for Combinatorial Auctions*, in Applied Optimization: Computational Methods in Decision-Making, Economics and Finance, E. J. Kontoghiorghes, B. Rustem and S. Siokos (editors), Kluwer Academic Publishers, pp. 455-479, September 2002.

2. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

3. 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.

4. V. Bafna, P. Berman and T. Fujito. *Constant ratio approximation of the weighted feedback vertex set problem for undirected graphs*, Int. Symp. on Algorithms and Computation, LNCS 1004, 1995, pp. 142-151.

5. A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. S. Naor, and B. Schieber. *A unified approach to approximating resource allocation and scheduling*, Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, 2000, pp. 735–744.

6. R. Bar-Yehuda. *One for the price of two: a unified approach for approximating covering problems*, Algorithmica, 27 (2), pp. 131-144, 2000.

7. R. Bar-Yehuda and S. Even. *A local-ratio theorem for approximating the weighted vertex cover problem*, Annals of Discrete Mathematics, 25, 1985, pp. 27-46.

8. R. Bar-Yehuda, M. M. Halldörsson, J. Naor, H. Shachnai and I. Shapira. *Scheduling split intervals*, 14th ACM-SIAM Symposium on Discrete Algorithms, pp. 732-741, 2002.

9. 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.

10. P. Berman and B. DasGupta. *A Simple Approximation Algorithm for Nonoverlapping Local Alignments (Weighted Independent Sets of Axis Parallel Rectangles)*, in Biocomputing, Volume 1, Panos M. Pardalos and Jose Principe (editors), Kluwer Academic Publishers, pp. 129-138, June 2002.

11. 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.

12. P. Berman and B. DasGupta. *Multi-phase Algorithms for Throughput Maximization for Real-Time Scheduling*, Journal of Combinatorial Optimization, Vol. 4, No. 3, September 2000, pp. 307-323.

13. P. Berman, P. Bertone, B. DasGupta, M. Gerstein, M.-Y. Kao and M. Snyder. *Fast Optimal Genome Tiling with Applications to Microarray Design and Homology Search*, Journal of Computational Biology, Vol. 11, No. 4, pp. 766-785, July 2004.

14. P. Berman, P. Bertone, B. DasGupta, M. Gerstein, M.-Y. Kao and M. Snyder. *Fast Optimal Genome Tiling with Applications to Microarray Design and Homology Search*, 2nd International Workshop on Algorithms in Bioinformatics (WABI 2002), LNCS 2452, R. Guigó and D. Gusfield (editors), Springer Verlag, pp. 419-433, July 2002.

15. P. Berman, B. DasGupta and S. Muthukrishnan. *Simple Approximation Algorithm for Nonoverlapping Local Alignments*, 13th ACM-SIAM Symposium on Discrete Algorithms, January 2002, pp. 677-678.

16. P. Berman, B. DasGupta, and S. Muthukrishnan. Slice and dice: A simple, improved approximate tiling recipe. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 455–464, January 2002.

17. P. Berman, B. DasGupta, and S. Muthukrishnan. On the exact size of the binary space partitioning of sets of isothetic rectangles with applications. *SIAM Journal of Discrete Mathematics*, 15 (2): 252-267, 2002.

18. P. Berman, B. DasGupta, S. Muthukrishnan, and S. Ramaswami. Improved approximation algorithms for tiling and packing with rectangles. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, 427–436, January 2001.

19. M. Chlebík and J. Chlebíková. *Approximation Hardness of Optimization Problems in Intersection Graphs of d-dimensional Boxes*, proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 267-276, 2005.

20. T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. *Introduction to Algorithms*, The MIT Press, 2001.

21. M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 635–644, January 2002.

22. K. A. Frenkel. The human genome project and informatics, *Communications of the ACM*, 34 (11), pp. 41-51, 1991.

23. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman, 1979.

24. D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge Univ Press, 1997.

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

26. D. Hochbaum. *Approximation Algorithms for NP-hard Problems*, PWS publishers, 1996.

27. D. Joseph, J. Meidanis and P. Tiwari. *Determining DNA sequence similarity using maximum independent set algorithms for interval graphs*, 3rd Scandinavian Workshop on Algorithm Theory, LNCS 621, pp. 326-337, 1992.

28. J. Jurka. Repbase Update: a database and an electronic journal of repetitive elements. *Trends in Genetics*, 9:418–420, 2000.

29. S. Khanna, S. Muthukrishnan, and M. Paterson. On approximating rectangle tiling and packing. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 384–393, 1998.

30. S. Khanna, S. Muthukrishnan, and S. Skiena. Efficient array partitioning. In G. Goos, J. Hartmanis, and J. van Leeuwen, editors, *Lecture Notes in Computer Science 1256: Proceedings of the 24th International Colloquium on Automata, Languages, and Programming*, 616–626. Springer-Verlag, New York, NY, 1997.

31. E. S. Lander, R. Langridge and D.M. Saccocio. *Mapping and interpreting biological information*, Communications of the ACM, 34 (11), pp. 33-39, 1991.

32. W. Miller, S. Scbwartz, and R. C. Hardison. *A point of contact between computer science and molecular biology*, IEEE Computational Science and Engineering, Spring 1994, pp. 69-78.

33. K. Mullis, F. Faloona, S. Scharf, R. Saiki, G. Horn, and H. Erlich. Specific enzymatic amplification of DNA in vitro: the polymerase chain reaction. *Cold Spring Harbor Symposium in Quantitative Biology*, 51:263–273, 1986.

34. S. Muthukrishnan, V. Poosala, and T. Suel. On rectangular partitions in two dimensions: Algorithms, complexity and applications. In *Proceedings of the 7th International Conference on Database Theory*, 236–256, 1999.

35. National Center for Biotechnology Information (NCBI). www.ncbi.nlm.nih.gov, 2002.

36. C. H. Papadimitriou. *Computational Complexity*, Addison-Wesley; reading, MA, 1994.

37. J. L. Rinn, G. Euskirchen, P. Bertone, R. Martone, N. M. Luscombe, S. Hartman, P. M. Harrison, K. Nelson, P. Miller, M. Gerstein, S. Weissman, and M. Snyder.

The transcriptional activity of human chromosome 22. *Genes and Development*, to appear.

38. W. L. Ruzzo and M. Tompa. Linear time algorithm for finding all maximal scoring subsequences. In *Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology*, 234–241, 1999.

39. A. F. A. Smit and P. Green. RepeatMasker, repeatmasker.genome.washington.edu, 2002.

40. A. Smith and S. Suri. Rectangular tiling in multi-dimensional arrays. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, 786–794, 1999.

41. T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

42. V. Vazirani. *Approximation Algorithms*, Springer-Verlag, July 2001.

43. J. C. Venter *et al. The sequence of the human genome*, Science, 291, pp. 1304-1351, 2001.

44. M.S. Waterman. *Sequence alignments*, in Mathematical Methods for DNA Sequences, M.S. Waterman (ed.), CRC, Boca Raton, FL, 1989, pp. 53-92.

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

46. Z. Zhang, P. Berman, and W. Miller. Alignments without low-scoring regions. *Journal of Computational Biology*, 5(2):197–210, 1998.