DIMACS Technical Report 2003-05
February 2003

Fast Optimal Genome Tiling with Applications to Microarray Design and Homology Search[1]

by

Piotr Berman[2] Paul Bertone[3] Bhaskar DasGupta[4]
Mark Gerstein[5] Ming-Yang Kao[6] Michael Snyder[7]

[1]A preliminary version of this paper appeared in the 2$^{\text{nd}}$ *Workshop on Algorithms in Bioinformatics*, Lecture Notes in Computer Science 2452, R. Guigó and D. Gusfield (editors), Springer Verlag, pp. 419-433, 2002.

[2]Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802. Email: `berman@cse.psu.edu`. Supported in part by National Library of Medicine grant LM05110.

[3]Department of Molecular, Cellular, and Developmental Biology & Department of Molecular Biophysics and Biochemistry, Yale University, New Haven, CT 06520. Email: `paul.bertone@yale.edu`. Supported in part by NIH grants P50 HG02357 and R01 CA77808.

[4]Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607. Email: `dasgupta@cs.uic.edu`. Supported in part by NSF grants CCR-0296041 and CCR-0208749, and a UIC startup grant.

[5]Department of Molecular Biophysics and Biochemistry & Department of Computer Science, Yale University, New Haven, CT 06520. Email: `mark.gerstein@yale.edu`. Supported in part by NIH grant P50 HG02357.

[6]Department of Computer Science, Northwestern University, Evanston, IL 60201. Email: `kao@cs.northwestern.edu`. Supported in part by NSF grant EIA-0112934.

[7]Department of Molecular, Cellular and Developmental Biology & Department of Molecular Biophysics and Biochemistry, Yale University, New Haven, CT 06520. Email: `michael.snyder@yale.edu`. Supported in part by NIH grants P50 HG02357 and R01 CA77808.

ABSTRACT

In this paper we consider several variations of the following basic tiling problem: given a sequence of real numbers with two size bound parameters, we want to find a set of tiles of maximum total weight such that each tiles satisfies the size bounds. A solution to this problem is important to a number of computational biology applications such as selecting genomic DNA fragments for PCR-based amplicon microarrays and performing homology searches with long sequence queries. Our goal is to design efficient algorithms with linear or near-linear time and space in the normal range of parameter values for these problems. For this purpose, we first discuss the solution to a basic online interval maximum problem via a sliding window approach and show how to use this solution in a non-trivial manner for many of the tiling problems introduced. We also discuss NP-hardness results and approximation algorithms for generalizing our basic tiling problem to higher dimensions. Finally, computational results from applying our tiling algorithms to genomic sequences of five model eukaryotes are reported.

1 Introduction

There are currently over 800 complete genome sequences available to the scientific community, representing the three principal domains of life: bacteria, archaea, and eukaryota [19]. The recent sequencing of several large eukaryotic genomes, including the nematode *Caenorhabditis elegans* [27] (100 Mb), the flowering plant *Arabidipsis thaliana* [26] (125 Mb), the fruitfly *Drosophila melanogaster* [1] (180 Mb) and working draft sequences for the laboratory mouse *Mus musculus* [28] (2.7 Gb) and *Homo sapiens* [11, 29] (3.4 Gb) has enabled advances in biological research on an unprecedented scale.

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 [23] which performs local subsequence alignments [25] against a database of known repetitive elements [13]. 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 paper.

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. A number of large-scale techniques have recently been developed for genome-wide data acquisition. Of these, a variety of microarray technologies have become ubiquitous in genome analysis centers due to the highly parallel nature of these experiments. DNA microarray experiments rely on the property of *complementarity*, whereby each of the four nucleotide bases can preferentially interact with another base: adenine (A) couples with thymine (T) and vice versa, and cytosine (C) couples with guanine (G) and vice versa. When nucleic acid molecules recognize and anneal to molecules having complementary sequences, they are said to *hybridize*. DNA microarrays exploit this phenomenon by measuring the degree of hybridization that takes place when a battery of different sequences is exposed to a set of test molecules or *probes*.

Each microarray element consists of a different DNA sequence, present in high copy number and immobilized to a glass surface. These DNA sequences are simultaneously exposed to a population of fluorescence-labeled probe molecules, and complementary sequences are allowed to hybridize. A laser scanner is used to excite the fluors coupled to the hybridized probe, and the emitted signals are detected and quantified by densitometry software. This provides a measure of the degree of hybridization of probe molecules to each microarray element. Since the DNA sequences immobilized to the array are known, this technique provides information about which sequences are enriched by complementary probe molecules that are present under various experimental conditions. Such data is used to observe the differential expression of genes across different cell types, identify the location of regulatory elements, and detect disease-related sequence mutations. Using microarray systems, researchers can simultaneously interrogate thousands of individual molecules in a single experiment. Designing microarrays that contain sequences representing large spans of chromosomal DNA provides a vehicle for the global analysis of gene expression or other types of molecular interactions on a genome-wide scale.

Various types of microarrays have been developed, each designed to capture a different kind of information. The most important types include cDNA microarrays [22], high-density oligonucleotide systems [16] and microarrays whose elements are composed of amplified genomic DNA [10, 12]. The principal differences between these systems lie in the substrate DNA molecules that are used. cDNA microarrays are constructed using gene sequences which have been previously been observed to be expressed within a cell. High-density oligonucleotide systems employ an *in situ* DNA synthesis method to deposit short (25-70 nucleotide) DNA fragments using a modified inkjet technology. These typically represent sequences internal to known genes, providing an anchor to which complementary probe molecules can hybridize.

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) [17] 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 [20].

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.

The rest of the paper is organized as follows. In Section 2 we precisely describe variants of the tiling problem. In Section 3 we discuss a basic Online Interval Maximum problem and its efficient solution via a sliding window approach. In Sections 4–8, we introduce new algorithms for the solution of various tiling problems. Finally, in Section 9 we discuss practical results for the tiling problem as applied to several example genome sequences.

2 Problem Statements and New Algorithms

In this section, we define our tiling problems, precisely describe algorithms for their solution, and cite related previous results. We also outline typical parameter values for our tiling problems when applied to DNA microarray design and homology search in Section 2.3.

2.1 Problem Statements

Based on the applications discussed in Section 1, 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 the basic genome tiling algorithm developed in [6], 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*[1]. 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 $i \in [1, m]$;
- each element of $c[S_{2j}, S_{2j+1})$ is x for all j;
- each element of $c[S_{2j-1}, S_{2j})$ is $-x$ for all j.

We note in Section 2.3 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 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 [4, 5, 14, 15, 18][2]. 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. To simplify exposition, unless otherwise stated explicitly, the GTile problem

[1] 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.

[2] 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 [14].

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.

2.2 Related Work

Tiling an array of numbers in one or more dimensions under various constraints is a very active research area (for example, see [3–5, 14, 15, 18, 24]) 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 [4], the shifting technique [9, Chapter 9] and dynamic programming methods based on binary space partitions [3, 14, 18] have proven useful for these problems. However, to the best of our knowledge, the particular tiling problems that we investigate in this paper have not specifically been looked at before except in [6] in which the authors provided a dynamic programming algorithm of higher time complexity to solve the basic GTile problem and some of its variations. Our problems are different from the tiling problems in [3, 4, 14, 15, 18, 24]; 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 [21] and [30]. The authors in [21] provide an $O(n)$ time algorithm to find all *maximal* scoring subsequences of a sequence of length n. In [30] the authors investigate computing maximal scoring subsequences which contain no subsequences with weights below a particular threshold.

2.3 Typical Parameter Values for Microarray Design and Homology Search

$n + 1$ (the DNA sequence length): Although the sizes of sequenced eukaryotic genomes range from 12Mb (for the budding yeast *Saccharomyces cerevisiae*) to 3.4Gb (*H. sapiens*), these exist as separate chromosomes that are treated as individual sequence databases by our tiling algorithms. Eukaryotic chromosomes range in size from approximately 230 Kb (*S. cerevisiae* chromosome I) to 256 Mb (human chromosome 1), with the average human chromosome being 150 Mb in size.

ℓ and u (lower and upper bounds for tile sizes): In computing an optimal set of tiles for microarray design, tile sizes can range from 200 bp to 1.5 Kb. Sequence fragments below 200 bp become difficult to recover when amplified in a high-throughput setting. An upper bound of 1.5 Kb balances two factors: (1) obtaining maximal sequence coverage with a limited number of tiles, and (2) producing a set of tiles which are small enough to achieve sufficient array resolution. In practice the average tile size is 800 when ℓ and u are set to 300 and 1500, respectively. For some instances of the homology search problem it may be desirable to extend the upper bound from 1.5 Kb to 2 Kb, representing the typical size of processed eukaryotic messenger RNA transcripts.

p **(maximum overlap between two tiles):** For microarray applications, tiles are disjoint; that is, the overlap parameter p is 0. However, searching sequence databases for homology matches can be enhanced by introducing a maximum overlap of $p \leq 100$ nucleotides for the case when potential matches can be made at tile boundaries.

t **(maximum number of tiles, when the number of tiles is bounded):** In selecting tiles for microarray applications, t can be specified to limit the number of sequence fragments considered for PCR amplification. For mammalian DNA where repeat content (and subsequent sequence fragmentation) is high, we can expect the high-complexity sequence nucleotides to cover $n/2$ sequence elements; the desired number of tiles to be computed will thus be $\frac{n}{2}$ divided by $\frac{u+\ell}{2}$ (the average of u and ℓ). For homology search problems t is unbounded.

m **(size of compressed input):** It is difficult to give an accurate estimate of the number of high-complexity sequence fragments in the target sequence following repeat screening since it varies greatly with the organism. In our experience, human chromosomes end up having between 2 to 3 times as many high-complexity sequence fragments (before processing) as there are final tiles (after processing), that is, m is roughly between $2t$ and $3t$. In other words, in practice m may be smaller than n by a factor of at least 600 or more.

2.4 Synopsis of Results

Our main results are summarized in Table 1; all these results are either new or direct improvements of any previously known algorithms. 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 the solution of an *Online Interval Maximum* (OLIM) problem. In Section 3, we discuss the OLIM problem together with an efficient solution for it using a windowing scheme reminiscent of that in [8]. However, the primary consideration in the applications in [8] 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 [8] since we need to maintain multiple windows of different sizes and data may not arrive at evenly spaced time intervals.

3 Computing the Online Interval Maximum via Sliding Window

In this section, we discuss an Online Interval Maximum (OLIM for short) problem which is used to design many of our remaining algorithms. The authors in [8] considered a restricted

Version of GTile	Time $O()$	Space $O()$	Approximation Ratio	Theorem
basic	n	n	exact	2
overlap is from a s-subset of $[0,\delta]$, $\delta < \frac{\ell}{2}$	sn	n	exact	4
compressed input	$m\frac{\ell}{u-\ell}$	$m\frac{\ell}{u-\ell}$	exact	5
number of tiles given	$\min\{n\log\frac{n}{\ell}, nt\}$	n	exact	7
d-dimensional	$\left(\left(\frac{u}{\ell}\right)\varepsilon\right)^{4\left(\frac{u}{\ell}\right)^2\varepsilon^2} M\varepsilon^2$	M	$\left(1-\frac{1}{\varepsilon}\right)^d$	16
d-dimensional, number of tiles given	$tM + dM\log^{\varepsilon} M$ $+dN\frac{\log N}{\log\log N}$	M	$\left(\prod_{i=1}^{d-1}\left(\lfloor 1+\log n_i\rfloor\right)\right)^{-1}$	16
	$M^{(2^{\varepsilon}-1)^{d-1}+1}\ dt$	$M^{(2^{\varepsilon}-1)^{d-1}+1}\ dt$	$\left(\prod_{i=1}^{d-1}\left(\lfloor 1+\frac{\log n_i}{\varepsilon}\rfloor\right)\right)^{-1}$	16

Table 1: A summary of the results in this paper. 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, $M = \prod_{i=1}^{d} n_i(u_i - \ell_i + 1)$, $N = \max_{1\le i\le d} n_i$ 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.

version of the OLIM problem in the context of maintaining stream statistics in the sliding window model and briefly mention a solution for this problem. We expect the OLIM problem to be useful in other contexts as well. The problem in its most general form can be stated as follows.

Input: (1) a sequence $a[0, n)$ of real values in increasing order where each value a_i is an *argument* or a *test* (possibly both), (2) 2α real numbers $\ell_1, u_1, \ell_2, u_2, \ldots, \ell_\alpha, u_\alpha$ with $0 < \ell_1 < u_1 < \ell_2 < u_2 < \cdots < \ell_\alpha < u_\alpha$ and (3) a real *value* function g defined on the *arguments.*

Output: for every test number a_k compute the maximum b_k of the α quantities $b_{k,1}, b_{k,2}, \ldots, b_{k,\alpha}$, where $b_{k,i}$ is given by

$$b_{k,i} = \max \left\{ g(a_j) : \ a_k - u_i \le a_j < a_k - \ell_i \text{ and } a_j \text{ is an argument} \right\}.$$

Online limitations: read the elements of the sequence $a[0, n)$ one at a time from left to right and compute b_k (if a_k is a test) *before* computing $g(a_k)$.

Theorem 1 *The OLIM problem can be solved in* $O(n_1\beta + n\alpha)$ *time using* $O(n_1 + \alpha)$ *space, where* n_1 *and* n_2 *are respectively the numbers of arguments and tests in the input and* β *is the maximum time to compute* $g(x)$ *for any* x*.*

Proof. We maintain a queue Q_i for each $i \in [1, \alpha]$. When we compute the pair $(a_k, g(a_k))$ for each argument a_k, we will also store it in the abovementioned queues such that the following invariant is satisfied for each Q_i: Q_i stores a *minimal* set of argument-value pairs such that for some future test a_m that has not been read yet it is possible to have $b_{m,i} = g(x)$ for some $(x, g(x))$ in Q_i. After reading each a_k, Q_i can be maintained using the following two rules:

Rule 1: Remove from Q_i every $(x, g(x))$ such that $x < a_k - u_i$. The validity for this rule is obvious from the definition of $b_{m,i}$ and the fact that the sequence $a[0, n)$ is in increasing order.

Rule 2: Let p be the *smallest index* of an argument such that $a_k - u_i \le a_p < a_k - \ell_i$ and $(a_p, g(a_p)) \notin Q_i$. Remove from Q_i every $(x, g(x))$ such that $g(x) \le g(a_p)$ and then insert $(a_p, g(a_p))$ in Q_i. Rule 2 is valid because for $m \ge k$ if we compute $b_{m,i}$ as the maximum value of a set that contains a removed $(x, g(x)$, then this set must also contain $(a_p, g(a_p))$. This is true because $x < a_p$ and therefore rule 1 would remove $(x, g(x))$ earlier.

If we perform all the needed insertions to Q_i using Rule 2, then the following holds: if $j < m$ and $(a_j, g(a_j))$ and $(a_m, g(a_m))$ are simultaneously present in Q_i, then $g(a_j) > g(a_m)$. Consequently, the maximum of the g-values in Q_i is contained in the *oldest* pair in Q_i. These observations allow us to maintain each Q_i as a double-ended queue where `front`(Q_i) stores the maximum of all the g-values of the elements in Q_i (needed to compute $b_{m,i}$, and to perform Rule 1), while `tail`(Q_i) has the minimum g-value (needed to perform Rule 2). The following is a high-level pseudocode of the main parts of our proposed algorithm for Q_i when reading a_k assuming all the parameters have been appropriately intialized:

```
(* windowing scheme for Qi *)
(* notations *)
Let q be the least index of an argument of a that has been read
    but has not considered for insertion to Qi yet;
for each argument ai, g(ai) is calculated once and stored;

(* algorithm for Qi *)
(* currently read number is ak *)

(* Execute Rule 1 *)
(x, g(x))← front(Qi);
while ((Qi ≠ ∅) and (x < ak − ui)
    remove front(Qi)
endwhile;

(* Execute Rule 2 *)
while ((ap < ak − ℓi))
    (y, g(y))← tail(Qi)
    while ((Qi ≠ ∅) and (g(y) ≤ g(ap))
        remove tail(Qi)
    endwhile;
    add (ap, g(ap)) to Qi at its tail;
    p← p + 1
endwhile;

(* calculate bk,i if necessary *)
if (ak is a test) then (z, g(z))← front(Qi); bk,i = g(z)
```

For each queue Q_i and each a_k we need to check if Q_i must be updated using either of the two rules. This takes $O(n\alpha)$ time. Because each argument is inserted to and deleted from a queue exactly once, these updates takes $O(n_1\alpha)$ total time. For every test we compute the maximum of the maxima of the queues, this takes $O(n_2\alpha)$ time. ❑

4 Basic GTile

Theorem 2 *The GTile problem can be solved in* $O(n)$ *time using* $O(n)$ *space.*

Proof. We use dynamic programming to reduce the GTile problem to OLIM. Subproblem k has $c[0,k)$ as input. Let m_k be the sum of weights of the tiles and $c[d_k, e_k)$ be the last tile in an optimum solution of subproblem k. If $m_k = m_{k-1}$, then subproblem k has the same solution as subproblem $k-1$, otherwise this solution consists of tile $c[d_k, k)$ and the tiles in the solution of subproblem d_k. Let $s_k = w(c[0,k))$, hence $w(c[i,j)) = s_j - s_i$. It is trivial to compute s_k for all $k \in [0, n]$ in $O(n)$ time and space. Obviously, $m_k = 0$ for $0 \leq k \leq \ell$. For $k > \ell$ we can compute m_k and d_k recursively as follows:

let $i \in [k - u, k - \ell] \cap [0, \infty)$ be an index that maximizes $v_i = m_i + s_k - s_i$;
if $v_i > m_{k-1}$, then $m_k = v_i$, $d_k = i$ and $e_k = k$
else $m_k = m_{k-1}$, $d_k = d_{k-1}$ and $e_k = e_{k-1}$

To complete our claim on time and space complexity, it suffices to show how to compute m_k for every $k > \ell$ in a total of $O(n)$ time and space. For each k we can first search for $i \in [k - u, k - \ell] \cap [0, \infty)$ that maximizes $y_i = m_i - s_i$; then we know $v_i = y_i + s_k$. This is the OLIM problem with input array $a[0, n)$, $a_i = i$, each a_i is both an argument and a test, $\alpha = 1$, $\ell_1 = \ell + 1$, $u_1 = u$ and $g(a_i) = m_i - s_i$. It is easy to recover an optimal tiling via the d_k and e_k values. ❑

5 GTile with Overlaps

In this section, we consider the GTile problem when the overlap p between two tiles is an element of some s-subset A of $[0, \delta]$ with $\delta < \frac{\ell}{2}$. The constraint $\delta < \frac{\ell}{2}$ holds for our biological applications since typically $p \leq 100$ and $\ell \simeq 300$. An important consequence of this constraint is the following observation:

Obsevration 3 *No c_i can belong to more than two tiles.*

Proof. Suppose that some c_i belongs to three tiles $c[b_1, e_1)$, $c[b_2, e_2)$ and $c[b_3, e_3)$ with $e_1 \leq e_2 \leq e_3$ and $b_3 < e_1$. Since each tile is of length at least ℓ and the length of overlap between any two tiles is less than $\frac{\ell}{2}$, we must have $b_1 \leq b_2 \leq b_3$. Now, $e_2 - b_2 = (e_2 - e_1) + (e_1 - b_2) < (e_2 - b_3) + (e_1 - b_2) < \frac{\ell}{2} + \frac{\ell}{2} = \ell$, hence the tile $c[b_2, e_2)$ does not satisfy the size bounds. ❑

Using the above observation, we can prove the following result.

Theorem 4 *The GTile problem with overlaps as described above can be solved in $O(sn)$ time using $O(n)$ space.*

Proof. Let $A \subseteq [0, p]$ be the s-subset for this problem. We reuse some of the notations in the proofs of Theorems 1 and 2. Let m'_k be the sum of weights of the tiles and $c[d'_k, k)$ was the last tile in an optimum solution of subproblem k in which c_k was the ending of the last tile. For $k \leq \ell$, $m'_k = 0$. For $k > \ell$, we can again compute m'_k and d'_k in the following manner:

let $i \in [k - u, k - \ell] \cap [0, \infty)$ be an index that
maximizes $v_i = m_i + s_k - s_i$;
$m'_k = v_i$, $d'_k = i$

This can again be solved in $O(n)$ time and space via the OLIM problem in the same manner as in Theorem 2. Let $[h_k, f_k)$ be tile previous to the last tile $[d_k, k)$ in an optimum solution of subproblem k in which c_k was the ending of the last tile. If the last tile was overlapped by the previous to last tile by $a \in A$ elements in an optimum solution of subproblem k, then the total weight of the solution is $(s_k - s_{d_k}) + m'_{d_k+a} - (s_{d_k+a} - s_{d_k}) = s_k + m'_{d_k+a} - s_{d_k+a}$. Now, for each $k > \ell$, we can compute m_k, d_k, e_k and f_k in the following manner:

(* ν_i is the solution when the last tile $[i, k)$ was not overlapped *)
let $i \in [k-u, k-\ell] \cap [0, \infty)$ be an index that maximizes $\nu_i = m_i + s_k - s_i$

(* $\mu_{j,a}$ is the solution when the last tile $[j, k)$ was overlapped by a elements *)
(* computation of $\mu_{j,a}$ takes $O(s)$ time *)
let $j \in [k-u, k-\ell] \cap [0, \infty)$ and $a \in A$ be the indices that maximize $\mu_{j,a} = s_k + m'_{j+a} - s_{j+a}$;

if $m_{k-1} \geq \max\{\nu_i, \mu_{j,a}\}$ then $m_k = m_{k-1}$, $d_k = d_{k-1}$, $e_k = e_{k-1}$, $f_k = f_{k-1}$
else
  if $\nu_i > \mu_{j,a}$ then $m_k = \nu_i$, $d_k = i$, $e_k = k$, $f_k = e_i$
  else $m_k = \mu_{j,a}$, $d_k = j$, $e_k = k$, $f_k = j + a$

By Observation 3, the last three tiles in the solution of subproblem k cannot have a common element and hence the solution is correct. Obviously, it suffices to find that $i \in [k-u, k-\ell] \cap [0, \infty)$ that maximizes $m_i - s_i$ and that $j \in [k-u+a, k-\ell+a] \cap [0, \infty)$ that maximizes $m'_j - s_j$. Each of them is can again be solved via OLIM problem. The tiles in an optimal solution can be recovered recursively via the d_i, e_i and f_i values. ❑

6 GTile with Compressed Input

Theorem 5 *The GTile problem with compressed input data can be solved in* $O(\alpha m)$ *time using* $O(\alpha m)$ *space where* $\alpha = \lceil \ell/(u-\ell) \rceil$.

In the remaining part of this section, we prove the above theorem. We again reduce this problem to OLIM. A key idea in this reduction is to *extend* the definition of a tile to include unions of adjacent tiles. In the rest of this section, we use this *extended definition* of a tile[3]. Then we can redefine our problem by requesting that tiles in our optimal solution are separated by entries that do not belong to any tile. Notice that the set of lengths that an extended tile may have is $\cup_{i=1}^{\infty}[i\ell, iu] = \left(\cup_{i=1}^{\alpha-1}[i\ell, iu]\right) \cup [\alpha\ell, \infty)$, where α is the smallest positive integer such that $(\alpha+1)\ell \leq \alpha u + 1$. We will say that the *legal length* of any tile in any solution can be of α different kinds: the i^{th} kind (for $1 \leq i < \alpha$) is $[i\ell, iu]$ and the α^{th} kind is $[\alpha\ell, \infty)$. The following lemma shows how to further restrict the beginnings and endings of our extended tiles.

Lemma 6 *There is an optimum solution for the GTile problem with compressed input in which the (extended) tiles are only of the following kinds:*

[3] Given an extended tile, it is indeed easy to decompose it into a set of tiles in $O(1)$ time.

(i) $c[S_{2i}, S_{2j+1})$, i.e., *starting and ending with a full block of x's;*

(ii) $c[S_{2i+1} - k\ell, S_{2i+1})$ *or* $c[S_{2i+1} - ku, S_{2i+1})$ *for some positive integer* $k \in [1, \alpha]$, i.e., *ending with a full block of x's and the length of the tile is either a multiple of ℓ or a multiple u;*

(iii) $c[0, k\ell)$ *or* $c[0, ku)$ *for some positive integer* $k \in [1, \alpha]$.

Proof. If the length of a selected tile is neither a multiple of ℓ nor a multiple of u, then it must be of type **(i)** since otherwise we could get a tile of greater weight by moving one of the ends. Now suppose that the length of a selected tile is a multiple of ℓ or u and it does not end at some S_{2i+1}. If it ends with a x, we can shift the tile to the right until it ends with the last x of this block of x's, or it coalesces with another tile in the solution and then the shifting process is again applied to this coalesced tile. If it ends with a $-x$, we can shift it to the right with similar results. This type of shifting produces tiles of type **(ii)** unless after shifting the tile to the left it ends at c_0 and then we get tiles of type **(iii)**. If the length of a tile of type **(ii)** or **(iii)** produced by the above shifting is greater than αu, then there are two possibilities: if the tile starts in a block of x's then we extend the tile until the begining of this block producing a tile of type **(i)**; however if the tile ends in a block of $-x$'s then we can shrink the tile by moving its begining to the right until either the length is αu or the tile ends with a full block of x's, whichever occurs earlier. ❑

In our algorithm, we first need to generate all possible beginnings and endings for the extended tiles. We do this in the following straightforward manner:

- We generate the monotonically increasing sequence of indices of *all possible beginnings* of our tiles: S_{2i} such that $2i \le m$ (type **(i)** and **(iii)**) and $S_{2i+1} - kL$ such that $0 < k \le \alpha$ and $L \in \{\ell, u\}$ (type **(ii)**). The total number of such beginnings is at most $\lceil \frac{m}{2} \rceil + \alpha m$. Duplicates are removed from the list.

- We generate the monotonically increasing sequence of indices of *all possible endings* of our tiles: S_{2j+1} such that $2j + 1 \le m$ (type **(i)** and **(ii)**) and kL such that $0 < k \le \alpha$ and $L \in \{\ell, u\}$. The total number of such endings is at most $\lceil \frac{m}{2} \rceil + 2\alpha$. Duplicates are removed from the list.

- We merge these monotonically increasing sequences of beginnings and endings into one increasing sequence `X` of beginnings and endings.

The above steps can be carried out in $O(m\alpha)$ time and space. As in Theorem 2, define s_i to be $w(c[0, i))$ for $i \in \{S_0, S_1, \ldots, S_m\}$. It is easy to calculate all such s_i's in $O(m)$ time and space. We now process the list `X` from left to right. We will maintain the following as we process each element x of the list `X`:

(a) If x is a beginning, then we maintain m_x, $v_x = m_x - s_x$ and $\mathsf{previous}(x) = c[d_x, e_x)$, where m_x the maximum possible sum of entries of set of disjoint tiles of legal length that end *before* x and $\mathsf{previous}(x)$ is the last tile from such a solution. If z is the entry in `X` before x, then we need to set $m_x = m_z$, $v_x = m_z - s_x$ and $\mathsf{previous}(x) = \mathsf{previous}(z)$; if x is the first element of `X` then we set $m_x = 0$ and $v_x = s_x$.

(b) If x is an ending, then we find $v_z = y \in \left(\cup_{i=1}^{\alpha-1}[x - i\ell, x - iu]\right) \cup [x - \alpha\ell, 0]\{v_y\}$. We set $m_x = (s_x - s_z) + m_z = s_x + v_z$, $v_x = m_x - s_x$ and $\mathtt{previous}(x) = c[c_x, d_x) = c[z, x)$. Let x' be the entry in X previous to x (if such an entry does not exist, then simply set $m_{x'} = -\infty$ in the following formula). Then, if $m_x < m_{x'}$ then we set $m_x = m_{x'}$.

It is clear that the only nontrivial step is the computation of m_z in **(b)** above. This is however again the OLIM problem: the sequence $a[0, n)$ is the sorted list of beginnings and endings X= $(x_1, x_2, \ldots)$, each beginning is an argument, an ending is a test, $g(x) = v_x$ for argument x, $[\ell_i, u_i] = [i\ell + 1, iu]$ for $1 \le i < \alpha$ and $[\ell_\alpha, u_\alpha] = [\alpha\ell + 1, n]$ Our final solution has a total weight of m_n; to recover the tiling we start with $\mathtt{previous}(m_n)$ in our collection and recursively look at the $\mathtt{previous}$ values for the part of the optimal solution that ends before the beginning of $\mathtt{previous}(m_n)$.

Because we have $O(m\alpha)$ arguments, $O(m + \alpha)$ tests and α queues, a straightforward application of Theorem 1 shows that we use $O(m\alpha^2)$ time and $O(m\alpha)$ space. However, we can design our algorithm with $O(m\alpha)$ running time as explained below.

For ease of counting in the analysis, we will assume that duplicates are not removed from the list of begining; this can only increase the running time. For analysis, the beginings will be partitioned into two sets: Γ_1 contains the $O(m)$ beginings of type S_{2i+1} and Γ_2 contains the remaining $m\alpha$ beginings. The endings are also partitioned into two sets: Δ_1 contains the 2α endings of the form kL for $0 < k \le \alpha$ and $L \in \{\ell, u\}$ and Δ_2 contains the remaining $O(m)$ endings.

First, let us calculate the total time that we will take to check if one of the α queues in the OLIM problem need to be updated.

(a) The $m\alpha$ beginings in Γ_2 can be partitioned into groups of 2α beginings, where each such group of 2α beginings are of the form $S_{2i+1} - kL$ for $0 < k \le \alpha$, $L \in \{\ell, u\}$ and a distinct S_{2i+1}. However, notice that the two endings $S_{2i+1} - kL$ for a particular k need to be checked for insertion in only the queue Q_k in the algorithm for the OLIM problem. With this minor modification in the code of the algorithm for the OLIM problem, the total time to check if all appropriate queues need to be updated for the beginings in Γ_2 is $O(m\alpha)$.

(b) For each begining in Γ_1 and each ending in Δ_2, we may need to check each of the α queues for update, hence the total time taken for these endings in $O(m\alpha)$.

(c) For a pair of endings $kL \in \Delta_1$, we need to check only one possible entry v_0 for insertion into its k^{th} queue Q_k; the remaining queues need not be checked for update for these pair of endings. Hence, the total time taken for the endings in Δ_1 is $O(\alpha)$.

Let us now calculate the total time taken for insertions and deletions of the beginings in the queues. Since each of the $m\alpha$ beginings in Γ_2 can be inserted and/or deleted from at most one queue, the total time for all these insertions and deletions is $O(m\alpha)$. Each of the remaining $O(m)$ beginings can be inserted or deleted from each of the α queues at most once, hence the total time taken for this part is also $O(m\alpha)$.

7 GTile with Bounded Number of Tiles

In this section, we consider the case when the maximum number of tiles t is given.

Theorem 7 *The GTile problem with bounded number of tiles can be solved in* $O(\min\{n \log n, nt\})$ *time using* $O(n)$ *space.*

It is not too difficult to provide an algorithm with $O(nt)$ time and space using the approach of Theorem 2 and maintaining separate queues for each possible value of number of tiles. In the rest of this section, we will use a different approach to reduce the space to $O(n)$ which is significant since t could be large. We will also provide another algorithm that runs in $O(n \log n)$ time using $O(n)$ space which is significant since typically for our applications $\log n \ll t$.

7.1 Sets and Sequences of Block Ends

Recall that a *block* is contiguous subsequence $c[p, q)$ of the given input sequence, a block of length at least ℓ and at most u is a tile and our solution consists of a set of disjoint tiles. A set of blocks $\mathbf{S}$ can be uniquely characterized by the set of endpoints of its blocks by using the following two quantities (where the first component of an ordered pair is λ or ρ depending on whether the endpoint is the left or the right endpoint of the block, respectively):

$$\mathit{ends}(c[a,b)) \; = \; \{(\lambda, a), (\rho, b)\}; \quad \mathit{ends}(\mathbf{S}) \; = \; \bigcup_{T \in \mathbf{S}} \mathit{ends}(T).$$

A *block end* $e = (\rho, m)$ has *side* $\mathit{side}(e) = \rho$ and *position* $\mathit{pos}(e) = m$. A set of ends E is *consistent* if $E = \mathit{ends}(\mathbf{S})$ for some set of non-empty blocks $\mathbf{S}$. We introduce a partial order $\prec$ among the block ends as follows: $e \prec f$ if $\mathit{pos}(e) < \mathit{pos}(f)$ or if $\mathit{pos}(e) = \mathit{pos}(f)$, $\mathit{side}(e) = \rho$ and $\mathit{side}(f) = \lambda$. A set E of $m+1$ ends ordered according to $\prec$ is the sequence $\overrightarrow{E} = (e_0, e_1, \ldots, e_m)$.

The test for consistency of E is obvious: the number of endpoints $m+1$ in E has to be even and the sequence $\mathit{side}(e_0)$, $\mathit{side}(e_1)$, $\ldots$, $\mathit{side}(e_m)$ has to be $(\lambda, \rho, \ldots, \lambda, \rho)$. Our insistence that $(\rho, k) \prec (\lambda, k)$ reflects the fact that we do not allow empty blocks, *i.e.* blocks of the form $c[k, k)$.

In this subsection we will assume that $\mathbf{S}$ and $\mathbf{T}$ are sets of blocks with $A = \mathit{ends}(\mathbf{S})$ and $A' = \mathit{ends}(\mathbf{T})$; hence both A and A' are consistent. We also assume that $B = A \oplus A' = (A - A') \cup (A' - A)$, $C = A \cup A'$ and $\overrightarrow{B} = (b_0, \ldots, b_{2k-1})$.

If $A \oplus D$ is consistent, we will say that D is an *alteration* of $\mathbf{S}$, and $\mathbf{S} \oplus D$ is the set of blocks $\mathbf{U}$ such that $\mathit{ends}(\mathbf{U}) = A \oplus D$. Obviously, B is an alteration of $\mathbf{S}$. We want to characterize the subsets of B that are alterations as well. For every $i \in [0, k)$ we say that b_{2i} and b_{2i+1} are *partners* in B. See Figure 1 for an illustration.

Lemma 8 *Partners in* B *are adjacent in* $\overrightarrow{C}$.

Proof. For the sake of contradiction, suppose that in $\overrightarrow{C}$ entries b_{2i} and b_{2i+1} are separated by another entry, say a, not in B. Then a is preceded by an odd number of elements of

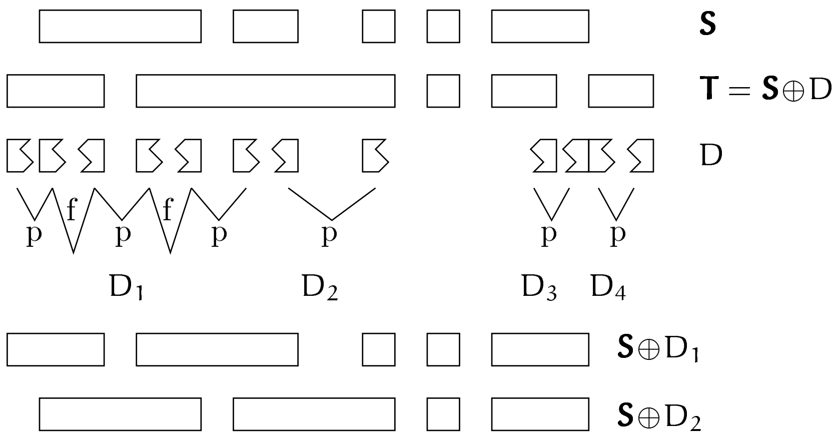

Figure 1: Alterations, partners and friends. A set of tiles is altered with a set of tile ends. Partners are indicated with p and friends with f. D_i's are atomic alterations.

B. Consequently, if a is preceded by an odd (respectively, even) number of elements of A, then it is preceded by an even (respectively, odd) number of elements of $A \oplus B$. Thus if the consistency of A dictates that $\mathit{side}(a) = \lambda$ (respectively, $\mathit{side}(a) = \rho$) then the consistency of $A \oplus B$ dictates that $\mathit{side}(a) = \rho$ (respectively, $\mathit{side}(a) = \lambda$), a contradiction. ❑

Lemma 9 *Assume that* $D \subset B$ *does not separate any pair of partners of* B*,* i.e. *for each pair of partners in* B*,* D *either has either both or none of them. Then* $A \oplus D$ *is consistent.*

Proof. Each $a \in A - D$ is preceded by an even number of elements of D, because if it is preceded by some $b \in D$ then, by Lemma 8 and the fact that D does not separate any pair of partners in B, it is also preceded by the partner of b which is also in D. Thus the parities of the positions of a in A and $A \oplus D$ are the same. Because $B - D$ also does not separate any pairs of partners in B and $A \oplus D = A' \oplus (B - D)$, the same reasoning shows that for any $d \in D - A \subset A' - (B - D)$ the parities of the positions of d in A' and $A \oplus D$ are the same. As a result, in the ordering of $A \oplus D$ every λ is on an even position and every ρ is on an odd position. ❑

7.2 Modifying a Set of Tiles

We now revise the assumptions of the previous subsection to assume that $\mathbf{S}$ and $\mathbf{T}$ are two sets of tiles (*i.e.* they satisfy the size bounds), and we redefine the notion of alteration as follows: D is an *alteration* of $\mathbf{S}$ if $\mathbf{S} \oplus D$ is a set of tiles. Again, we want to characterize the alterations of $\mathbf{S}$ that are subsets of B.

If $g < h < i < j$ and $c[g,i],\ c[h,j] \in \mathbf{S} \cup \mathbf{T}$ we say that (λ, h) and (ρ, i) are friends; see Figure 1 for an illustration. We need the following lemma.

Lemma 10 *Two friends must be adjacent in* $\overrightarrow{C}$*, they must both belong to* B *and they are not partners.*

Proof. Without loss of generality assume that $c[g,i] \in \mathbf{S}$ and $c[h,j] \in \mathbf{T}$. Clearly, $(\lambda,g) \prec (\lambda,h) \prec (\rho,i) \prec (\rho,j)$. No block end in A is between (λ,g) and (ρ,i) and no block end in A' is between (λ,h) and (ρ,j). This shows that (λ,h) and (ρ,i) are adjacent in $\overrightarrow{C}$ and that they are both in B. To see that they are not partners, note that $A \oplus \{(\lambda,h),(\rho,i)\}$ cannot be consistent and use Lemma 9. ❏

Note that a pair of friends is easy to recognize: it must be a pair of the form $\{b_{2i-1}, b_{2i}\} = \{(\lambda,g),(\rho,h)\}$ where either $b_{2i-1} \in A - A'$ and $e_{2i} \in A' - A$, or $b_{2i-1} \in A' - A$ and $b_{2i} \in A - A'$, Let G_B be the graph with the vertex set as the set of block ends B and with two kinds of edges: between pairs of partners and between pairs of friends. By Lemmas 8 and 10 these sets of edges form two *disjoint* matchings of G_B. Now we have our crucial lemma.

Lemma 11 *If* $D \subseteq B$ *is the set of vertices in a connected component of* G_B*, then* D *is an alteration of* $\mathbf{S}$*.*

Proof. Because D does not separate any pair of partners, by Lemma 9 $\mathbf{U} = \mathbf{S} \oplus D$ is a set of disjoint blocks. Suppose that $c[g,h] \in \mathbf{U}$ is not a tile, *i.e.* $h - g \notin [\ell, u]$; we will then obtain a contradiction. Obviously $c[g,h] \notin \mathbf{S}$ and $c[g,h] \notin \mathbf{T}$. Hence exactly one of (λ,g) and (ρ,h) is in A and the other one is in A'. Without loss of generality assume that $c[g,i] \in \mathbf{S}$ and $c[j,h] \in \mathbf{T}$. We can exclude the cases when $j \le g < i \le h$ or $g \le j < h \le i$ because then $h - g \in [\ell, u]$. The case when $g < j < i < h$ can also be excluded, because then the friends (λ,j) and (ρ,i) are separated by D (because D contained (ρ,i) but did not contain (λ,j)) and hence D did not contain all the vertices in a connected component of G_B. Similarly, if $j < g < h < i$ then the friends (λ,g) and (ρ,h) are separated by D. Thus it remains to consider the case when $g < i \le j < h$. But then $(\rho,i) \in D$, $(\lambda,j) \notin D$, $(\rho,h) \in D$. This contradicts the assumption that D is a connected component of G_B. ❏

Alterations that are vertices in a connected component of G_B will be called *atomic*; see Figure 1 for an illustration. Obviously any alteration can be expressed as a union of one or more disjoint atomic alterations and two disjoint atomic alterations can be applied in any order on a given set of tiles to obtain the same set of tiles. We will say that an atomic alteration is *increasing, neutral* or *decreasing* if $|\mathbf{S} \oplus D| - |\mathbf{S}|$ equals 1, 0 or -1, respectively. See Figure 2 for an illustration.

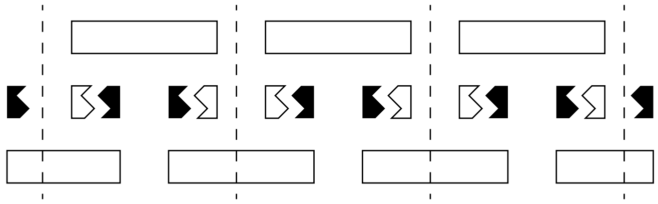

Figure 2: An increasing atomic alteration. White tile ends are old and black tile ends are new. There can be also an increasing atomic alteration that does not have the first two tile ends or the last two: this is the case when ends of new and old tiles coincide.

Lemma 12 *If* D *is an atomic alteration of* $\mathbf{S}$*, then* $-1 \le |\mathbf{S} \oplus D| - |\mathbf{S}| \le 1$.

Proof. Except for its first and last elements, D is covered by disjoint pairs of friends. Applying a pair of friends removes one element of $\mathit{ends}(\mathbf{S})$ and inserts another, thus leaving the total number of block ends unchanged. Hence the net change in the number of block ends can come only from the first and the last elements of D, and because the number of block ends changes by at most 2, the number of blocks changes by at most 1. ❑

7.3 Computing S_t in $O(nt)$ Time Using $O(n)$ Space

Let $\mathbf{S}_0 = \emptyset$ and $\mathbf{S}_{t+1}$ be a set of $t + 1$ tiles of *maximum* weight that can be obtained by applying a *minimal* alteration (*i.e.* an alteration that is not properly contained in another alteration) to $\mathbf{S}_t$.

Lemma 13 *If* $\mathbf{S}_{t+1} = \mathbf{S}_t \oplus D$ *then* D *is an atomic alteration.*

Proof. Suppose that D is *not* an atomic alteration and thus it is a union of more than one *disjoint* atomic alterations. We know that disjoint atomic alternations can be applied in any order. If one of them, say D_0, is neutral, then one of the following two cases occur:

- D_0 changes the weight of the set of tiles. Then $w(\mathbf{S}_t \oplus D_0) > w(\mathbf{S}_t)$ or $w(\mathbf{S}_{t+1} \oplus D_0) > w(\mathbf{S}_{t+1})$, contradicting the definition of $\mathbf{S}_t$ or $\mathbf{S}_{t+1}$.
- D_0 does not change the weight of the set of tiles. Then we could apply $D - D_0$ rather than D to get $\mathbf{S}_{t+1}$, *i.e.* D is not minimal.

If D contains a decreasing atomic alteration D_0, then it must also contains an increasing alteration, say D_1, and we can use the neutral alteration $D_0 \cup D_1$ to obtain a contradiction similar to above. Hence every atomic alterations contained in D is increasing and thus D can contain only one such alteration. ❑

Based on the above results, our simple algorithm is as follows.

```
S_0 = ∅, w(S_0) = 0
for p = 1 to t do
    compute S_p = S_{p-1} ⊕ D by finding the increasing atomic
      alteration D that produces maximum gain in total weight
    if w(S_p) ≤ w(S_{p-1}) then
       output S_{p-1}, exit (* S_p is the best solution *)
output S_t
```

Our claim on the time and space complexity of the above algorithm follows from the following lemma.

Lemma 14 *Given* S_{p-1}*, we can find in* $O(n)$ *time and space an atomic alteration* D *such that* $S_p = S_{p-1} \oplus D$.

Proof. For simplicity, we will be using $\mathbf{S}$ to denote $\mathbf{S}_{p-1}$. As in Theorem 2, we compute $s_k = w(c[0,k))$ for all $k \in [0,n]$ in $O(n)$ time and space.

Let us consider possible structures of the atomic alteration D that we seek. For a possible tile end e let $D_e = \{f \in D : f \prec e\}$. Note that D_e does not include e itself. In our algorithm we will scan all possible tile ends in $\prec$ order from left to right. In other words, we will scan the elements of $c[0,n)$ in left-to-right order, and each element will be first considered as a possible tile ending and then as a possible tile begining. When we consider the tile end e we need to know what D_e could be. We need to perform the following case analysis. We refer the reader to Figure 2 for a clearer understanding of these cases. The default value of the quantity $A(i)$ defined below is $-\infty$ for all $i \in [0,n)$.

Case 1: $e = (\lambda, i)$.

Case 1.1: for some $c[g,h) \in \mathbf{S}$ we have $g < i < h$.

It follows that $D_e \neq \emptyset$ since otherwise D would not be an increasing atomic alteration and that $E = D_e \cup \{(\rho,h)\}$ is a neutral alteration. Let $\mathbf{A}(i)$ be the set of neutral atomic alterations that contain (ρ,h) as the only element that does not precede e and let

$$A(i) = \max_{E \in \mathbf{A}(i)} w(\mathbf{S} \oplus E) - w(\mathbf{S})$$

Because $w(\mathbf{S})$ cannot be increased by a neutral atomic alteration, $A(i) \leq 0$. Later when we need to consider e as a possible left end of a new tile, we will consider it with a priority of $\nu_i = A(i) - s_i$.

Case **2.1** shows how to compute the value of $A(i)$ when the last new tile ends at i using the OLIM problem. Note that since any element is first considered as a possible ending and then a possible ending, Case **2** will occur before Case **1** for that element. Hence, we simply set $A(i)$ to be the maximum of $A(i-1)$ and current $A(i)$, and update ν_i accordingly if necessary.

Case 1.2: Case **1.1** does not hold. Then e would have to be the first tile end in D and thus the left end of the leftmost new tile. We should now consider e as a possible left end with priority $\nu_i = -s_i$. We should allow here for e being the left end of an old tile (*i.e.* $i = g$ for some $c[g,h) \in \mathbf{S}$) which indicates an alteration where the leftmost new tile shares its left end with an old tile.

Case 2: $e = (\rho, i)$.

Case 2.1: for some $c[g,h) \in \mathbf{S}$ we have $g < i < h$.

We consider e to be the right end of a new tile. The matching left end, (λ, j), must satisfy $j \leq g$, *i.e.* it must be to the left of $c[g,h)$. We choose j as the index of the matching left end that satisfies $\nu_j = \max_{i-u \leq z \leq \min\{i-\ell, g\}}\{\nu_z\}$, and this allows to consider a new alteration to the set $\mathcal{A}(i)$; the total gain of this alteration is $\nu_j - w(c[g,h)) + s_i$ Then, if this gain is greater than the current value of $A(i)$, we replace $A(i)$ with this new gain and update ν_i accordingly. Notice that computation of the ν_j's can be done via the OLIM problem in the same manner as in the proof of Theorem 2.

Case 2.2: Case **2.1** does not hold and $c[g, h)$ is the last tile of $\mathbf{S}$ to the left of i. Then e would be the rightmost tile end of an increasing atomic alteration (or, if $i = h$, the right end of a tile whose left end is the rightmost element of the alteration). We select the matching left end (λ, q) such that $\nu_q = \max_{i-u \leq z \leq i-\ell} \nu_z$ and the gain of this alteration is $s_i + \nu_q$. Then, if this gain is greater than the current value of $A(i)$, we replace $A(i)$ with this new gain and update ν_i accordingly. Notice that the computation of the ν_q's can again be done via the OLIM problem.

❑

Remark 1 *The claim of Lemma 14 holds even if* D *is a neutral or decreasing alteration (which produces a minimum total weight loss) by using a very similar algorithm. Moreover we actually also compute, for each possible tile end* e*, the optimum alteration of the prescribed type in which all elements precede* e*.*

7.4 Computing S_t in $O(n \log \frac{n}{\ell})$ Time Using $O(n)$ Space

The idea of this algorithm is the following. We will proceed in phases. Before a phase, we computed a set of t disjoint tiles, say $\mathbf{S}$, that has the *largest* weight under the constraint that each tile is contained in one of the blocks $c[a_0, a_1), c[a_1, a_2), \ldots, c[a_{k-1}, a_k)$. For this phase, we will select some a_i such that, after the phase, we replace $\mathbf{S}$ with some $\mathbf{S} \oplus B$ that maximizes the sum of weights under the constraint that each tile in $\mathbf{S} \oplus B$ is contained in $c[a_0, a_1)$, $c[a_1, a_2)$, $c[a_{i-2}, a_{i-1})$, $c[a_{i-1}, a_{i+1})$, $c[a_{i+1}, a_{i+2}), \ldots, c[a_{k-1}, a_k)$ (*i.e.* the new set of blocks is obtained from the old set of blocks by coalescing the two blocks $c[a_{i-1}, a_i)$ and $c[a_i, a_{i+1})$ into one block $c[a_{i-1}, a_{i+1})$).

We can start with $a_i = i\ell$; each block is a tile of minimum length, thus we can compute the weight of each tile and select the tiles with t largest weights. Using any linear time algorithm for order statistics [7], we can complete the first phase in $O(n)$ time and space. In Lemma 15 below we show that we can perform a single phase in $O(M + \log \frac{n}{\ell})$ time and $O(n)$ space, where $M = \max_{i=1}^{k}\{a_i - a_{i-1}\}$. This will be sufficient for our claim on the time and space complexity of our complete algorithm by the following analysis[4]. We first coalesce adjacent pairs of blocks of length ℓ into blocks of length 2ℓ (unless there is only one block of length ℓ left). This requires $O(n/\ell)$ phases and each of them takes $O(\ell)$ time, because during these phases the longest block has length 2ℓ. Hence the total time and space complexity for these n/ℓ phases is $O(n)$. Repeating the same procedure for blocks of length $2\ell, 4\ell, \ldots$, it follows that that if all blocks but one have the maximum length then we can half the number of blocks in $O(n)$ time and space, and again, all blocks but one will have the maximum length. Obviously, we are done when we have one block. Since each phase can be carried out independently of any other phase, the space complexity is $O(n)$, and the total time complexity is $O\left(\sum_{i=1}^{\log(n/\ell)} \left(\left(2^{i+1}\ell + \log \frac{n}{\ell}\right) \frac{n}{2^i \ell}\right)\right) = O\left(n \log \frac{n}{\ell}\right)$. Hence it suffices to prove the following lemma to prove our claim.

Lemma 15 *We can perform a single phase in* $O(M + \log n)$ *time and* $O(n)$ *space.*

[4]This is similar to the analysis of mergesort in which two blocks of sorted numbers are merged during one step [7].

Proof. For an increasing (respectively, decreasing, neutral) atomic alteration B, let B be called the *best* increasing (respectively, decreasing, neutral) atomic alteration if $w(\mathbf{S} \oplus B)$ is maximized. We maintain the following data structures throughout all phases. For each block, we store that part of the current solution that is contained in that block, the best increasing atomic alteration of that part and the best decreasing one. Moreover, we will have two priority queues Q_1 and Q_2 of blocks, in which the *priority* of a block is the *gain* in total weight of its best increasing and decreasing atomic alterations, respectively. These data structures can be initialized in $O(n)$ time and space using Lemma 14 and Remark 1. Moreover, note that both Q_1 and Q_2 contain at most $O(n/\ell)$ entries.

We need to show how to find the desired alteration B which consists of an union of one or more atomic alterations. We consider all possible structures of B. If the new solution contains no tile $c[g, h)$ such that $g < a_i < h$, then $B = \emptyset$. Otherwise, assume that such a tile exists. Under this assumption, the number of end points of tiles contained in each of $c[a_{i-1}, a_i)$ and $c[a_i, a_{i+1})$ changes from even to odd, which shows that B contains a pair of partners b, b' such that $b \prec (\rho, a_i)$ and $(\lambda, a_i) \prec b'$. Let D be the atomic alteration such that $\{b, b'\} \subseteq D \subseteq B$. We have the following 3 cases.

Case 1: D is a neutral atomic alteration. Because $|\mathbf{S} \oplus (B - D)| = |\mathbf{S}|$, we could alter $\mathbf{S}$ with $B - D$ before the current phase, hence this alteration cannot increase the weight, and therefore we do not need it. Thus in this case $B = D$. Hence, for this case, we need to find a best neutral atomic alteration in $c[a_{i-1}, a_{i+1})$, *i.e.* one that yields the maximum increase of weight. Using Lemma 14 and Remark 1 we can find such an alteration in time $O(a_{i+1} - a_{i-1})$ and space. Subsequently, in similar time and space we can find in $c[a_{i-1}, a_{i+1})$ a best increasing and decreasing atomic alterations and update Q_1 and Q_2 in $O\left(\log \frac{n}{\ell}\right)$ time.

Case 2: D is an increasing atomic alteration. $C = B - D$ does not contain a non-empty atomic alteration F that does not change the size of $\mathbf{S}$; otherwise since we could alter $\mathbf{S}$ with F before the current phase, F cannot increase the weight of $\mathbf{S}$ and thus we do not need it. Hence C is a decreasing atomic alteration.

To find B, we consider two subcases.

Case 2.1: C is contained in a block different than $c[a_{i-1}, a_{i+1})$. We can find a best D using the algorithm from Lemma 14, in $O(M)$ time and a best C from the priority queue Q_2 in $O(n/\ell)$ time, both using $O(n)$ space. Subsequently, we need to compute a best increasing and a best decreasing atomic alterations for both $c[a_{i-1}, a_{i+1})$ and the block that contains C (using Lemma 14 and Remark 1) and update Q_1 and Q_2 in a total of $O\left(M + \log \frac{n}{\ell}\right)$ time and $O(n)$ space.

Case 2.2: C is contained in $c[a_{i-1}, a_i)$. We first compute for each $j \in [a_{i-1}, i)$ the best decreasing atomic alteration contained in $c[a_{i-1}, j)$ and set $X(j)$ to be its *gain* *i.e.* the increase in total weight produced by it (perhaps negative). As noted in Remark 1, this can be done in $O(M)$ time and $O(n)$ space. Now, to find the best increasing atomic alteration D, we proceed as in Lemma 14 except that when we consider some (λ, k) to be the leftmost left end of a tile introduced by D we set its priority to be $X(j) - s_k$. Hence such a D can again be found in $O(M)$ time and

$O(n)$ space. At the end, we must update Q_1 and Q_2 as in Case **1** in $O\left(\log \frac{n}{\ell}\right)$ time.

Case 2.3: C is contained in $c[a_i, a_{i+1})$. This case is mirror-symmetric of Case **2.2**, so we can apply the same methods, except that we will be scanning the merged block from right to left.

Case 3: D is a decreasing atomic alteration. Then $C = B - D$ is an increasing atomic alteration by an argument very similar to that in Case **2**. Case **3** is then symmetric of Case **2** if we replace increasing (respectively, decreasing) by decreasing (respectively, increasing) in the remainder of Case **2**.

❑

8 GTile in d-dimensions

It is not difficult to see from previously known results that the GTile problem is NP-hard even for $d = 2$. The following theorem summarizes approximability issues for the higher dimensional cases.

Theorem 16 *Let* $M = \prod_{i=1}^{d} n_i(u_i - \ell_i + 1)$, $N = \max_{1 \le i \le d} n_i$, $\frac{u}{\ell} = \max_i \frac{u_i}{\ell_i}$, *and* $\varepsilon > 1$ *be any arbitrary given constant. Then, it is possible to design the following approximation algorithms for the GTile problem in* d*-dimension:*

(i) *if the number of tiles is unbounded, then it is possible to design an* $O\left(\left(\left(\frac{u}{\ell}\right)\varepsilon\right)^{4\left(\frac{u}{\ell}\right)^2\varepsilon^2} M\varepsilon^2\right)$ *time algorithm using* $O(M)$ *space with an approximation ratio of* $\left(1 - \frac{1}{\varepsilon}\right)^d$.

(ii) *if the number of tiles is bounded, then approximation algorithms with the following bounds are possible:*

- *an* $O(tM + dM \log^{\varepsilon} M + dN \frac{\log N}{\log \log N})$ *time algorithm using* $O(M)$ *space with an approximation ratio of* $1/\left(\prod_{i=1}^{d-1}(\lfloor 1 + \log n_i \rfloor)\right)$.
- *an* $O(M^{(2^{\varepsilon}-1)^{d-1}+1}\, dt)$ *time algorithm using* $O(M^{(2^{\varepsilon}-1)^{d-1}+1}\ dt)$ *space with an approximation ratio of* $1/\left(\prod_{i=1}^{d-1}\left(\lfloor 1 + \frac{\log n_i}{\varepsilon} \rfloor\right)\right)$.

Proof. The GTile problem in d dimensions can be easily reduced to the d-RPACK problem in [5] in which the number of rectangles is M and the coordinates of the i^{th} dimension has coordinates from $\{1, 2, \ldots, n_i\}$. This gives us the results in **(ii)**. Since the i^{th} dimension of any tile has a length of at least ℓ_i and at most u_i, the aspect ratio of any tile is at most $\frac{u}{\ell}$ and hence we can use the shifting strategy of [9, Chapter 9] to get the result in **(i)**. ❑

9 Computational Results

Here we discuss 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 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.

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: 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.

The results of applying the tiling algorithm to each chromosome appear in Table 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.

GTile performs particularly well for the mammalian chromosomes, dramatically increasing the high-complexity sequence coverage with minimal repeat inclusion. The higher eukaryotic genomes benefit most from this approach, because the large number of repetitive elements present in these genomes results in a much more difficult tiling problem. Also notable is that as sequence coverage increases, the number of required tiles decreases as

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 3: 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.

a greater percentage of tiles are generated within the optimal size range. The algorithm therefore minimizes the number of tiles required to obtain the maximum coverage of high-complexity sequence.

10 Acknowledgements

We would like to thank Ms. Sarita Lella of the Bioengineering department in UIC for implementing the basic GTile algorithm.

References

[1] M. D. Adams et al. The genome sequence of Drosophila melanogaster. *Science*, 287:2185–2195, 2000.

[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] 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.

[4] 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.

[5] 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.

[6] P. Bertone, M. Y. Kao, M. Snyder, and M. Gerstein. The maximum sequence tiling problem with applications to DNA microarray design (submitted).

[7] T. H. Cormen, C. L. Leiserson and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[8] 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.

[9] D. S. Hochbaum. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing, Boston, MA, 1997.

[10] C. E. Horak, M. C. Mahajan, N. M. Luscombe, M. Gerstein, S. M. Weissman, and M. Snyder. GATA-1 binding sites mapped in the beta-globin locus by using mammalian chip-chip analysis. *Proceedings of the National Academy of Sciences of the U.S.A.*, 995:2924–2929, 2002.

[11] International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 15:860–921, 2001.

[12] V. R. Iyer, C. E. Horak, C. S. Scafe, D. Botstein, M. Snyder, and P. O. Brown. Genomic binding sites of the yeast cell-cycle transcription factors SBF and MBF. *Nature*, 409:33–538, 2001.

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

[14] 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.

[15] 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.

[16] D. J. Lockhart et al. Expression monitoring by hybridization to high-density oligonucleotide arrays. *Nature Biotechnology*, 14:1675–1680, 1996.

[17] 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.

[18] 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.

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

[20] 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* (in press).

[21] 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.

[22] D. D. Shalon and P. O. B. S. J. Smith. A DNA microarray system for analyzing complex DNA samples using two-color fluorescent probe hybridization. *Genome Research*, 6(7):639–645, July 1996.

[23] A. F. A. Smit and P. Green. RepeatMasker, repeatmasker.genome.washington.edu, 2002.

[24] 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.

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

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

[27] The C. elegans Sequencing Consortium. Genome sequence of the nematode C. elegans: a platform for investigating biology. *Science*, 282:2012–2018, 1998.

[28] The Mouse Genome Sequencing Consortium. Initial sequencing and comparative analysis of the mouse genome. *Nature*, 420:520–562, 2002.

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

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