# Fast Optimal Genome Tiling with Applications to Microarray Design and Homology Search

Piotr Berman[1], Paul Bertone[2], Bhaskar DasGupta[3], Mark Gerstein[4], Ming-Yang Kao[5], and Michael Snyder

[1] Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802
`berman@cse.psu.edu`
Supported in part by National Library of Medicine grant LM05110
[2] Department of Molecular, Cellular, and Developmental Biology, Yale University, New Haven, CT 06520
`paul.bertone@yale.edu`
Supported in part by NIH grants P50 HG02357 and R01 CA77808
[3] Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607 `dasgupta@cs.uic.edu`
Supported in part by NSF grant CCR-0296041 and a UIC startup fund
[4] Department of Molecular Biophysics and Biochemistry and Department of Computer Science, Yale University, New Haven, CT 06520
`mark.gerstein@yale.edu`
[5] Department of Computer Science, Northwestern University, Evanston, IL 60201
`kao@cs.northwestern.edu`
Supported in part by NSF grant EIA-0112934
[6] Department of Molecular, Cellular, and Developmental Biology and Department of Molecular Biophysics and Biochemistry, Yale University, New Haven, CT 06520
`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 such that they satisfy the size bounds and the total weight of the tiles is *maximized.* This solution to this problem is important to a number of computational biology applications, such as selecting genomic DNA fragments for amplicon microarrays, or 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 discuss the solution of 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 our tiling problems. We also discuss NP-hardness and approximation algorithms for generalization of our basic tiling problem to higher dimensions.

## 1 Introduction and Motivation

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* [26] (100 Mb), the flowering plant *Arabidipsis thaliana* [25] (125 Mb), the fruitfly *Drosophila melanogaster* [1] (180 Mb) and a working draft sequence for *Homo sapiens* [11, 27] (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 comprise families of transposable elements that can move from one chromosomal location to another, retroviral sequences integrated 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 these repeat sequences can be problematic for both computational and experimental biology research. **Homology searches** [2] of repeat-laden queries against large sequence databases often result in many spurious 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 molecules.

Genomic DNA can be screened for repeat sequences using specialized programs such as RepeatMasker [22] which performs local subsequence alignments [24] against a database of known repetitive elements [13]. Repeats are then masked within the query sequence, where 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, extremely 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 the subsequent sections of this paper.

Another motivation for looking at the tiling problems considered in this paper is their application to **DNA microarray designs** 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 couple 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 test molecule or *probe*. Each array element consists of immobilized DNA sequences which are exposed to a fluorescence-labeled probe. The signals emitted by the bound probe molecules are detected and quantified with a laser scanner. Using microarray systems, researchers can simultaneously interrogate thousands of individual molecules in a single experiment. Designing microarrays that contain sequences representing large spans of genomic DNA provides a vehicle for the global analysis of gene expression or other types of molecular interactions.

Various types of microarrays have been developed, each designed to capture a different kind of information. The most include cDNA microarrays [21], high-density oligonucleotide systems [16], and microarrays whose elements are composed of amplified genomic DNA [12, 10]. 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 may hybridize.

In contrast, the microarrays we consider here are composed of larger (typically 500bp-1.5 Kb) sequences of genomic DNA that are acquired via the *polymerase chain reaction* (PCR) [17], in which segments 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 genomic sequence.

A maximal-coverage amplicon microarray can be designed by deriving a tile path through a target genomic sequence such that the best set of tiles is selected for PCR amplification. We are therefore concerned with finding the maximum number of high-complexity subsequence fragments (tiles) given a genomic DNA sequence whose repetitive elements have been masked. Deriving 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 Definitions, Notations and Problem Statements

Based on the applications discussed in the previous section, we formalize a family of tiling problems as discussed below. These problems build upon a basic genome tiling problem in [6], which we call the *GTile problem* and describe as follows. For notational convenience, let $[i, j)$ denote integer set $\{i, i+1, \ldots, j-1\}$, $[i, j]$ denote $[i, j+1)$, and $f[i, j)$ denote the elements of an array $f$ with indices in $[i, j)$. Our inputs consist of an array $c[0, n)$ of *real* numbers and two size parameters $\ell$ 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 $[l, u]$ 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:** In the important special case of the GTile problem when all entries of $c[0, n)$ belong to $\{-x, x\}$ for some fixed $x$, the input sequence can be more efficiently represented by specifying blocks of identical values. In other words, we can compress the input $c[0, n]$ to an integer sequence $d[0, m]$ such that
- $S_0 = 0$, $S_{i+1} = S_i + d_i$, $S_m = n + 1$, where $d_0 = 0$ is allowed;
- 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 $j$.

Notice that the input size of such a compressed input data is $m$, which is typically much smaller than $n$ (see Section 2.2).

**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.

**No overlap versus overlap:** It may be useful to relax this condition by allowing two tiles to share at most $p$ elements, for some given (usually small) $p > 0$. However, we will penalize this overlap by subtracting the weight of an 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 $w(\mathcal{T}) - \sum_{c_i \in \mathcal{R}} c_i$.

**1-dimensional versus $d$-dimensional:** Generalization of the GTile problem in $d$ dimensions has potential applications in database designs. In this case, we are given a $d$-dimensional array $\mathbf{C}$ of size $n_1 \times n_2 \times \cdot \times n_d$ and $2d$ 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 again 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 as discussed before. To simplify exposition, unless otherwise stated explicitly, the GTile problem we consider is 1-*dimensional*, has an *uncompressed* input, an *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 each dimension for the $d$-dimensional input array, $d[0,m]$ is the sequence of $m+1$ integers for the compressed input data in 1-dimension, $w(\mathcal{T})$ 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. Many proofs are omitted due to page limitations; they can be found in the full version of the paper.

### 2.1 Related Work

Tiling an array of numbers in one or more dimensions under various constraints is a very active research area (for example, see [4,3,5,14,15,18,23]) 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 been looked at before. Our problems are different from the tiling problems in [4,3,14,15,18,23]; 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. The papers which most closely relate to our work are the references [20] and [28]. The authors in [20] provide an $O(n)$ time algorithm to find all *maximal* scoring subsequences of a sequence of length $n$. In [28] the authors investigate computing maximal scoring subsequences which contain no subsequences with weights below a particular threshold.

### 2.2 Typical Parameter Values for Microarray Design and Homology Search Applications

**$n+1$ (the DNA sequence length):** Although the sizes of sequenced eukaryotic genomes range from 12 Mb (for the budding yeast *Saccharomyces cere-*

*visiae*) to 3.4 Gb (*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.

**$\ell$ and $u$ (lower and upper bounds for tile sizes):** In computing an optimal tile path 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 $\ell$ and $u$ are set to 300 and 1500, respectively. For the homology search problem it is desirable to extend the upper bound from 1.5 Kb to 2 Kb, representing the average size of mammalian messenger RNA transcripts.

**$p$ (maximum overlap between two tiles):** For microarray applications, tiles are disjoint; that is, the overlap parameter $p$ is 0, and no sequence elements are shared between any two tiles. However, searching sequence databases for homology matches can be enhanced by introducing a maximum overlap of $p \leq 100$ nucleotides. This condition provides 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 $\ell$). 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.

### 2.3 Synopsis of Results

All our results are summarized in Table 1, except the result that GTile problem is NP-hard in two or more dimensions. Most of our algorithms use simple data structures (such as a double-ended queue) and are 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 a 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 the 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 needed 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.

| Version of GTile | Time | Space | Approximation Ratio | Theorem |
|---|---|---|---|---|
| basic | $n$ | $n$ | exact | 2 |
| overlap is 0 or $p < \ell/2$ | $n$ | $n$ | exact | 3 |
| compressed input | $\frac{\ell}{u-\ell}(m + \log \frac{\ell}{u-\ell})$ | $m$ | exact | 4 |
| number of tiles given | $\min\{n \log \frac{n}{\ell}, tn\}$ | $n$ | exact | 5 |
| $d$-dimensional | $\left(\left(\frac{u}{\ell}\right)\varepsilon\right)^{4\left(\frac{u}{\ell}\right)^2\varepsilon^2} M\varepsilon^2$ | $M$ | $\left(1 - \frac{1}{\varepsilon}\right)^d$ | 6 |
| $d$-dimensional number of | $tM + dM \log^{\varepsilon} M$ $+dN\frac{\log N}{log \log N}$ | $M$ | $\left(\Pi_{i=1}^{d-1}\left(\lfloor 1 + \log n_i \rfloor\right)\right)^{-1}$ | 6 |
| tiles given | $M^{(2^{\varepsilon}-1)^{d-1}+1}\ dt$ | $M^{(2^{\varepsilon}-1)^{d-1}+1}\ dt$ | $\left(\Pi_{i=1}^{d-1}\left(\lfloor 1 + \frac{\log n_i}{\varepsilon} \rfloor\right)\right)^{-1}$ | 6 |

**Table 1.** A summary of the results in this paper. The parameter $\varepsilon > 1$ is any arbitrary *constant.* For the $d$-dimensional case, $M = \Pi_{i=1}^{d} n_i(u_i - \ell_i + 1)$, $N = \max_{1 \leq i \leq d} n_i$ and $\frac{u}{\ell} = \max_i \frac{u_i}{\ell_i}$. For our biology applications $p \leq 100 < \frac{\ell}{2} \ll n$, $t \simeq \frac{n}{u+\ell}$, $m$ is much smaller than $n$ and $\frac{\ell}{u-\ell} < 6$. The column of Approximation Ratio indicates whether the algorithm computes the optimal solution exactly or, for an approximation algorithm, the ratio of our total weight to that of the optimum.

## 3 Solving An Online Interval Maximum Problem via Sliding Window Approach

In this section, we discuss the Online Interval Maximum (OLIM for short) problem, which is used to design many of our algorithms. The authors in [8] mentions a restricted version of the OLIM problem in the context of maintaining stream statistics in the *sliding window* model and briefly mentions 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) $2\alpha$ real numbers

$\ell_1, u_1, \ell_2, u_2, \ldots, \ell_\alpha, u_\alpha$ with $0 \leq \ell_1 < u_1 < \ell_2 < u_2 < \cdots < \ell_\alpha < u_\alpha$ and (3) a real function $g$ defined on the *arguments.*

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

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

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

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

*Proof.* In our algorithm we will maintain a queue $Q_i$ for each $(\ell_i, u_i)$. We will compute the pair $(a, g(a))$ for each argument $a$; these pairs will be also stored in the abovementioned queues with the following property: $Q_i$ stores a minimal set of argument-value pairs such that it is possible for some test $a_m$ that as yet has not been read we have $b_{m,i} = g(x)$ for some $(x, g(x))$ in $Q_i$. $Q_i$ can be maintained using the following two rules:

**Rule 1.** After reading $a_k$, remove from $Q_i$ every $(x, g(x))$ such that $x < a_k - u_i$.

**Rule 2.** Let $p$ be the smallest index of an argument such that $a_k - u_i \leq a_p < a_k - \ell_i$ and $(a_p, g(a_p)) \notin Q_i$. After reading $a_k$, remove from $Q_i$ every $(x, g(x))$ such that $g(x) \leq g(a_p)$. Then insert $(a_p, g(a_p))$ in $Q_i$.

Rule 2 is valid because for $m \geq k$ 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 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 show that we need to maintain $Q_i$ 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).

For each queue $Q_i$ of the $\alpha$ queues and each $a_k$ we need to check if the queue has to be updated using either of the two rules. This takes $O(n\alpha)$ time. Because each argument is inserted and deleted to a queue exactly once, the aggregate of these updates takes $O(n_1\alpha)$ 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 as input $c[0,k]$. Let $m_k$ be the sum of weights of the tiles and $d_k$ and $e_k$ be the left (beginning) and right (end) index of 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 $s[0,n)$ in $O(n)$ time and space. Obviously, $m_k = 0$ for $0 \leq k < \ell$. For $k \geq \ell$ we can compute $m_k$ and $d_k$ recursively:

```
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 prove our claim, it suffices to show how to compute such an index $i$ for every $k \in [\ell, n]$ in a total of $O(n)$ time and space. Equivalently, for each $k$ we can 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, $\alpha = 1$, $\ell_1 = \ell$, $u_1 = u$, and, for $0 \leq i < n$, $g(i) = m_i - s_i$.

It is also easy to recover an optimal tiling via the $d_i$ and $e_i$ values.

## 5 GTile with Restricted Overlap

In this section, we consider the GTile problem when either two tiles are disjoint or they have exactly $p$ elements in common, for some fixed $p < \ell/2$. The constraint $p < \ell/2$ is true for our biology applications since typically $p \leq 100$ and $\ell \simeq 300$.

**Theorem 3.** *The GTile problem with restricted overlap can be solved in $O(n)$ time using $O(n)$ space.*

*Remark 1.* Consider the GTile problem with overlap in which the overlap can be any integer from $[1,p]$ with $p < \ell/2$. An approach similar to above shows how to solve this problem in $O(pn)$ time using $O(pn)$ space.

## 6 GTile with Compressed Input

**Theorem 4.** *The GTile problem with compressed input data can be solved in $O(\alpha(m + \log \alpha))$ time using $O(m)$ space where $\alpha = \lceil \ell/(u-\ell) \rceil$.*

## 7 GTile with Bounded Number Of Tiles

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

**Theorem 5.** *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 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 also significant since $\log n$ is usually much smaller than $t$.

## 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 $\ell$ and at most $u$ is a tile and our solution consists of a set of disjoint tiles. A set of blocks $\boldsymbol{S}$ 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 $\lambda$ of $\mu$ depending on whether the endpoint is the left or the right endpoint of the block, respectively):

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

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

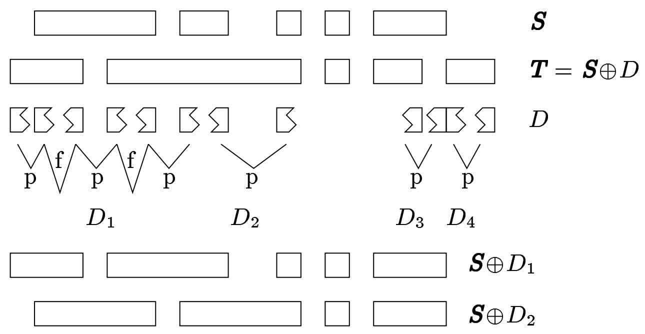
The test for consistency of $E$ is obvious: the number of endpoints $m$ in $E$ has to be even and the sequence $\mathit{side}(e_0)$, $\mathit{side}(e_1)$, …, $\mathit{side}(e_m)$ has to be $(\lambda, \rho, \ldots, \lambda, \rho)$. Our insistence that $(\rho, k) \prec (\lambda, k)$ reflects the fact that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

In this subsection we will assume that $\boldsymbol{S}$ and $\boldsymbol{T}$ are sets of blocks with $A = \mathit{ends}(\boldsymbol{S})$ and $A' = \mathit{ends}(\boldsymbol{T})$; 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 $\boldsymbol{S}$, and $\boldsymbol{S} \oplus D$ is the set of blocks $\boldsymbol{U}$ such that $\mathit{ends}(\boldsymbol{U}) = A \oplus D$. Obviously, $B$ is an alteration of $\boldsymbol{S}$. 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 1** *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 $B$. 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



$\boldsymbol{S}$
$\boldsymbol{T} = \boldsymbol{S} \oplus D$
$D$
f
f
p
p
p
p
p
p
$D_1$
$D_2$
$D_3$
$D_4$
$\boldsymbol{S} \oplus D_1$
$\boldsymbol{S} \oplus D_2$

**Fig. 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.

$\mathit{side}(a) = \lambda$ (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 2** *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.*

## 7.2 Modifying a Set of Tiles

We now slightly modify the assumptions of the previous subsection. Now we assume that $\boldsymbol{S}$ and $\boldsymbol{T}$ 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 $\boldsymbol{S}$ if $\boldsymbol{S} \oplus D$ is a set of *tiles*. Again, we want to characterize the alterations of $\boldsymbol{S}$ that are subsets of $B$.

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

**Lemma 3** *Two friends must be adjacent in $\overrightarrow{C}$, they must both belong to $B$ and they are not partners.*

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; see Figure 1 for an illustration. By Lemmas 1 and 3 these sets of edges form two disjoint matchings of $G_B$. Now we have our crucial lemma.

**Lemma 4** *If $D \subseteq B$ is the set of vertices in a connected component of $G_B$, then $D$ is an alteration of $\boldsymbol{S}$.*

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 $|\boldsymbol{S} \oplus D| - |\boldsymbol{S}|$ equals 1, 0 or $-1$, respectively. See Figure 2 for an illustration.

**Lemma 5** *If $D$ is an atomic alteration of $\boldsymbol{S}$, then $-1 \leq |\boldsymbol{S} \oplus D| - |\boldsymbol{S}| \leq 1$.*

## 7.3 Computing $\boldsymbol{S}_t$ in $O(nt)$ Time Using $O(n)$ Space

Let $\boldsymbol{S}_0$ be the empty set of tiles, and let $\boldsymbol{S}_{t+1}$ be a set of $t+1$ tiles of maximum weight, and, under this limitation, one that is obtained by applying a *minimal* alteration (*i.e.* an alteration that is not properly contained in another alteration) to $\boldsymbol{S}_t$.

**Lemma 6** *If $\boldsymbol{S}_{t+1} = \boldsymbol{S}_t \oplus D$ then $D$ is an atomic alteration.*

Our algorithm is straightforward. We start with $\boldsymbol{S}_0 = \emptyset$ and then, for $0 < p \leq t$ we compute $\boldsymbol{S}_p$ from $\boldsymbol{S}_{p-1}$ if the total weight of $\boldsymbol{S}_p$ if greater than that of $\boldsymbol{S}_p$ (otherwise we stop). By Lemma 6 we need to find an increasing atomic alteration that results in the maximum possible weight gain. Our claim on the time and space complexity of the algorithm follows from the following lemma.

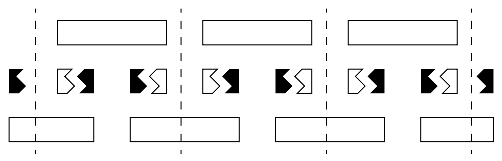


**Fig. 2.** An increasing atomic alteration. White tile ends are old, and black tile ends are new. There can be also an increasing atomic alteration that does not have the first two tile ends, or the last two: this is the case when ends of new and old tiles coincide.

**Lemma 7** *Given $S_{p-1}$, we can find in $O(n)$ time and space an atomic alteration $D$ such that $S_p = S_{p-1} \oplus D$.*

*Remark 2.* The claim of Lemma 7 holds even if $D$ is a neutral or decreasing alteration by using a very similar algorithm. Moreover, we actually 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 $\boldsymbol{S}$, 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 $\boldsymbol{S}$ with some $\boldsymbol{S} \oplus B$ that maximizes the sum of weights under the constraint that each tile in $\boldsymbol{S} \oplus B$ 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 8 below we show that we can perform a single phase in $O(M + \log n)$ 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[7]. We first coalesce adjacent pairs of blocks of length $\ell$ into blocks of length $2\ell$ (unless there is only one block of length $\ell$ left). This requires at most $n/\ell$ phases and each of them takes $O(\ell)$ time, because during these phases the longest block has length $2\ell$. Hence the total time and space complexity for these $n/\ell$ 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 only. Since each phase can be carried out independently of any other phase, the space complexity is $O(n)$, and the total time complexity is $O\left(\sum_{i=1}^{\log(n/\ell)} \left(\left(2^{i+1}\ell + \log n\right)\ \frac{n}{2^i \ell}\right)\right) = O\left(n \log \frac{n}{\ell}\right)$. Hence it suffices to prove the following lemma to prove our claim.

**Lemma 8.** *We can perform a single phase in $O(M + \log n)$ time and $O(n)$ space.*

## 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 6.** *Let $M = \Pi_{i=1}^{d} n_i(u_i - \ell_i + 1)$, $N = \max_{1 \le i \le d} n_i$, $\frac{u}{\ell} = \max_i \frac{u_i}{\ell_i}$, and $\varepsilon > 0$ and $c \ge 1$ be any two arbitrary given constants. Then, it is possible to design the following approximation algorithms for the GTile problem in d-dimension:*

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

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

**(2)** *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* $1/\left(\Pi_{i=1}^{d-1}\left(\lfloor 1+\log n_i\rfloor\right)\right)$.
- *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* $1/\left(\Pi_{i=1}^{d-1}\left(\lfloor 1+\frac{\log n_i}{\varepsilon}\rfloor\right)\right)$.

*Proof.* The GTile problem in $d$ dimension can be easily reduced to the $d$-RPACK problem in [5] in which the number of rectangles is $M$ and the coordinates of the $i^{\text{th}}$ dimension has coordinates from $\{1,2,\ldots,n_i\}$. This gives us the results in **(2)**. Since the $i^{\text{th}}$ dimension of any tile has a length of at least $\ell_i$ 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 **(1)**.

## 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 for journal publication.
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, H. Dong, M. C. Byrne, M. T. Follettie, M. V. Gallo, M. S. Chee, M. Mittmann, C. Wang, M. Kobayashi, and H. Horton 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. 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.
21. 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.
22. A. F. A. Smit and P. Green. RepeatMasker, repeatmasker.genome.washington.edu, 2002.
23. 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.
24. T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.
25. The Arabidipsis Genome Initiative. Analysis of the genome sequence of the flowering plant arabidopsis thaliana. *Nature*, 408:796–815, 2000.
26. The C. elegans Sequencing Consortium. Genome sequence of the nematode c. elegans: a platform for investigating biology. *Science*, 282:2012–2018, 1998.
27. J. C. Venter et al. The sequence of the human genome. *Science*, 291:1304–1351, 2001.
28. Z. Zhang, P. Berman, and W. Miller. Alignments without low-scoring regions. *Journal of Computational Biology*, 5(2):197–210, 1998.