Faster Optimal Algorithms For Segment Minimization With Small Maximal Value*

Technical Report CS 2011-08

Therese Biedl[1], Stephane Durocher[2], Céline Engelbeen[3], Samuel Fiorini[4], and Maxwell Young[1]

[1] David R. Cheriton School of Computer Science, University of Waterloo, ON, Canada
{biedl,m22young}@uwaterloo.ca
[2] Department of Computer Science, University of Manitoba, MB, Canada
durocher@cs.umanitoba.ca
[3] Département de Mathématique, Université Libre de Bruxelles, Brussels, Belgium
{celine.engelbeen,sfiorini}@ulb.ac.be

Abstract. The segment minimization problem consists of finding the smallest set of integer matrices (*segments*) that sum to a given intensity matrix, such that each summand has only one non-zero value (the *segment-value*), and the non-zeroes in each row are consecutive. This has direct applications in intensity-modulated radiation therapy, an effective form of cancer treatment.

We study here the special case when the largest value H in the intensity matrix is small. We show that for an intensity matrix with one row, this problem is fixed-parameter tractable (FPT) in H; our algorithm obtains a significant asymptotic speedup over the previous best FPT algorithm. We also show how to solve the full-matrix problem faster than all previously known algorithms. Finally, we address a closely related problem that deals with minimizing the number of segments subject to a minimum *beam-on-time*, defined as the sum of the segment-values. Here, we obtain a almost-quadratic speedup over the previous best algorithm.

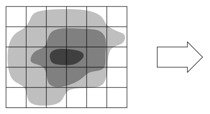
1 Introduction

Intensity-modulated radiation therapy (IMRT) is an effective form of cancer treatment, where radiation produced by a linear accelerator is delivered to the patient through a multileaf collimator (MLC). The MLC is mounted on an arm that can revolve freely around the patient so that he or she can be irradiated from several angles. We focus on the so-called *step-and-shoot* mode, where the radiation is delivered in a series of steps. In each step, two banks of independent metal leaves in the MLC are positioned to obstruct certain portions of the radiation field, while leaving others exposed. Neither the head of the MLC, nor its leaves move during irradiation. A treatment plan specifies the amount of radiation to be delivered along each angle.

For any given angle, the radiation field is discretized and decomposed into $m \times n$ pixels, where m is typically the number of pairs of leaves of the MLC. This determines a decomposition of the radiation beam into $m \times n$ *beamlets*. The amount of radiation is represented as an $m \times n$ *intensity matrix* A of non-negative integer values, whose entries represent the amount of radiation to be delivered through the corresponding pixel, along the given angle.

The leaves of the MLC can be seen as partially covering rows of A; for each row i of A there are two leaves, one of which may slide inwards from the left to cover the elements in columns 1 to $\ell - 1$ of that row, while the other may slide inwards from the right to cover the elements in columns $r + 1$ to n. Thus the entries of A that are not covered form an interval $[\ell, r] := \{\ell, \ell + 1, \dots, r\}$ of consecutive columns. After each step, the amount of radiation applied in that step (this can differ per step) is subtracted from each entry of A that has not been covered. The irradiation for the given angle is completed when all entries of A have reached 0.

* This work was supported by the "Actions de Recherche Concertées" (ARC) fund of the "Communauté française de Belgique", and the National Sciences and Engineering Research Council of Canada (NSERC). C.E. acknowledges support from the "Fonds pour la Recherche dans l'Industrie et l'Agriculture" (F.R.I.A.).



$$\begin{array}{|c|c|c|c|c|c|}\hline 0&1&1&1&1&0\\\hline 1&1&2&2&2&1\\\hline 1&2&4&4&2&0\\\hline 1&1&2&2&1&0\\\hline 0&1&1&0&0&0\\\hline\end{array} = 2 \times \begin{array}{|c|c|c|c|c|c|}\hline 0&0&0&0&0&0\\\hline 0&0&0&0&0&0\\\hline 0&0&1&1&1&0\\\hline 0&0&0&0&0&0\\\hline 0&0&0&0&0&0\\\hline\end{array} + 1 \times \begin{array}{|c|c|c|c|c|c|}\hline 0&1&1&1&0&0\\\hline 0&0&1&1&1&0\\\hline 0&1&1&1&0&0\\\hline 0&0&1&1&0&0\\\hline 0&0&0&0&0&0\\\hline\end{array} + 1 \times \begin{array}{|c|c|c|c|c|c|}\hline 0&0&0&0&1&0\\\hline 1&1&1&1&1&1\\\hline 1&1&1&1&0&0\\\hline 1&1&1&1&1&0\\\hline 0&1&1&0&0&0\\\hline\end{array}$$

Fig. 1. An example of a segmentation of an intensity matrix where $H = 4$.

Setting leaf positions in each step of the treatment plan requires time. Minimizing the number of steps reduces treatment time, which increases patient comfort, and can result in increased patient throughput, reduced machine wear, and overall reduced cost of the procedure. Minimizing the number of steps for a given treatment plan is the primary objective of this paper.

Formally, a *segment* is a $m \times n$ binary matrix S such that ones in each row of S are consecutive. Each segment S has an associated non-negative integer weight which we call the *segment-value*, denoted by $v_{\mathcal{S}}(S)$ or simply $v(S)$ when $\mathcal{S}$ is understood. We call a segment an t-segment if its value is t. A *segmentation* of A is a set of segments whose weighted sum equals A. So, $\mathcal{S}$ is a segmentation of A if and only if we have $A = \sum_{S \in \mathcal{S}} v(S)S$. Figure 1 illustrates the segmentation of an intensity matrix.

The *(minimum-cardinality) segmentation problem* is, given an intensity matrix A, to find a minimum cardinality segmentation of A. In this paper, we also consider the special case of a matrix A with one row, which we call the *single-row segmentation problem*, in contrast with the more general *full-matrix segmentation problem* which consists of m multiple rows.

Finally, as a consequence of our results on the segmentation problem, we also briefly examine a different, but closely related *lex-min* problem: find a minimum cardinality segmentation among those with minimum *beam-on-time*, defined as the total value $\sum_{S \in \mathcal{S}} v(S)$ of the segmentation.[4] As the segmentation problem focuses on the time incurred for establishing leaf positions, optimizing the beam-on-time also has implications for making procedures more efficient by reducing the time spent administering the treatment corresponding to the segments themselves.

Related Work

The segmentation problem is known to be NP-complete in the strong sense, even for a single row [9, 2, 3], as well as APX-complete [4]. Bansal *et al.* [4] provide a $24/13$-approximation algorithm for the single-row problem and give better approximations for more constrained versions. Work by Collins *et al.* [10] shows that the single-*column* version of the problem is NP-complete and provides some non-trivial lower bounds given certain constraints. Work by Luan *et al.* [16] gives two approximation algorithms for the full $m \times n$ segmentation problem, and Biedl *et al.* [6] extend this work to achieve better approximation algorithms that result in performance improvements.

A number of heuristics are known [3, 20, 11, 14] as well as approaches for obtaining optimal (exact) solutions [7, 1, 19]. Particularly relevant to our work is that of Cambazard *et al.* [8] who show that the segmentation of a single row is fixed-parameter tractable (FPT); specifically, they give an algorithm which achieves an optimal segmentation in $O(p(H)^2\, n)$ time, where H is the largest value in A and $p(H)$ is the number of partitions of H.

Kalinowski [15] studies the lex-min problem and gives polynomial time algorithms for when H is a constant. In the single-row case, he gives a $O(p(H)^2\, n)$ time algorithm. The solution output by this first algorithm is also optimal for the minimum-cardinality segmentation problem (this follows from known results, e.g. [4]). For general $m \times n$ intensity matrices, he provides a $O(2^H \sqrt{H}\, m\, n^{2H+2})$ time algorithm. From this second algorithm, one can derive an algorithm for the full $m \times n$ minimum segmentation problem with time complexity $O(2^H H^{5/2}\, m\, n^{2H+3})$. This is done by guessing the beam-on-time T of a minimum cardinality segmentation and appending a row to the intensity matrix to increase its minimum beam-on-time to T; it can be shown that $T \in O(H^2\, n)$.

Our Contributions

We summarize our contributions below:

[4] The lex-min problem is also known as the *min DT-min DC* problem where DT stands for *decomposition time* (i.e., the beam-on-time) and DC stands for decomposition cardinality (i.e., the number of segments); however, we refer to this as the lex-min problem throughout.

– For the single-row segmentation problem, we provide a faster exact algorithm. In particular, our algorithm runs in $O(p(H)\,H\,n)$ time, which is polynomial in n so long as $H \in O(\log^2 n)$. In comparison to the result of Cambazard *et al.* [8], our algorithms is faster by a factor of $\Omega(p(H)/H)$; to the best our knowledge, ours is the asymptotically fastest FPT algorithm.

Despite our result for the single-row problem, significant challenges remain in solving the full-matrix problem and here we achieve two important results:

– For general H, we give an algorithm that yields an optimal solution to the full-matrix segmentation problem in $O(m\,n^H/2^{(1-\epsilon)(H)})$ time for an arbitrarily small constant $\epsilon > 0$. Note, that with a minor change, the result of Kalinowski [15] can be applied to solve the segmentation problem (i.e. ignoring the aspect of beam-on time); however, the worst case running time is $\Omega(m\,n^{2H+2})$. Therefore, our result yields a better-than-quadratic improvement in the running time.
– For $H = 2$, the full matrix problem can be solved optimally in $O(m\,n)$ time in contrast to the $O(m\,n^2)$ time implied by the previous result for general H. This result also has implications for the approximation algorithms in [6] where it can be employed as a subroutine to improve results in practice.

Finally, we address the lex-min problem:

– For general H, we give an algorithm that yields an optimal solution to the full-matrix lex-min problem in time $O(m\,n^H/2^{(\frac{1}{2}-\varepsilon)H})$. In comparison to the previous best result by Kalinowski [15], our algorithm yields a almost-quadratic improvement in the running time and, to the best of our knowledge, obtains the fastest asymptotic time complexity to date.

Therefore, our algorithms represent a significant asymptotic speed-up and the techniques required to achieve these improvements is non-trivial. In the appendix, we specify the necessary data structures for our algorithms, along with some discussion of trade-offs between time and space complexity, which should be of use in practice.

2 Single-row segmentation

In this section, we prove that the single-row segmentation problem is FPT in H, the largest value in the intensity matrix A. Since A has only one row, we represent it as a vector $A[1..n]$. We call a segmentation of $A[1..n]$ *compact* if any two segments in it *begin* (i.e., have their first non-zero entries) at different indices, and *end* (i.e., have their last non-zero entries) at different indices. The following observation is straightforward; we give a proof in Section A of the appendix.

Lemma 1. *For any segmentation $\mathcal{S}$ of a single row, there exists a compact segmentation $\mathcal{S}'$ with $|\mathcal{S}'| \leq |\mathcal{S}|$.*

Our algorithm uses a dynamic programming approach that computes an optimal segmentation of any prefix $A[1..j]$ of A. We say that a segmentation of $A[1..j]$ is *almost-compact* if any two segments in it begin at different indices, and any two segments in it either end at different indices or both end at index j. We will only compute almost-compact segmentations; this is sufficient by Lemma 1. We compute the segmentation conditional on the values of the last segments in it.

Let $\mathcal{S}$ be a segmentation of vector $A[1..j]$; each $S \in \mathcal{S}$ is hence a vector $S[1..j]$. Define the *signature* of $\mathcal{S}$ to be the multi-set obtained by taking the value $v(S)$ of each segment ending in j. Note that the signature of a segmentation of $A[1..j]$ is a *partition* of $A[j]$, i.e., a multi-set of positive integers that sum to $A[j] \leq H$.

We briefly review some notation for multi-sets. A multi-set $\mathcal{M}$ with entries from the universe $[H] := \{1, \ldots, H\}$ can be described via the H-tuple $(m_1(\mathcal{M}), \ldots, m_H(\mathcal{M}))$, where $m_t(\mathcal{M})$ denotes the multiplicity of element t in $\mathcal{M}$. We use $||\mathcal{M}|| := \sum_{t=1}^{H} m_t(\mathcal{M})$ to denote the cardinality of $\mathcal{M}$. For two such multi-sets $\mathcal{M}_1$ and $\mathcal{M}_2$, let $\mathcal{M}_1 \cup \mathcal{M}_2$ be the multi-set defined by $m_t(\mathcal{M}_1 \cup \mathcal{M}_2) := m_t(\mathcal{M}_1) + m_t(\mathcal{M}_2)$ for $t \in [H]$, let $\mathcal{M}_1 \cap \mathcal{M}_2$ be the multi-set defined by $m_t(\mathcal{M}_1 \cap \mathcal{M}_2) := \min\{m_t(\mathcal{M}_1), m_t(\mathcal{M}_2)\}$ for $t \in [H]$, and let $\mathcal{M}_1 - \mathcal{M}_2$ be the multi-set defined by $m_t(\mathcal{M}_1 - \mathcal{M}_2) := \max\{0, m_t(\mathcal{M}_1) - m_t(\mathcal{M}_2)\}$ for $t \in [H]$. *Adding (resp. deleting)*

Notation	Definition
A	A $m \times n$ intensity matrix. For $m = 1$, $A[1..j]$ is a vector.
H	Largest value in A.
$[H]$	The set $\{1, \ldots, \mathrm{H}\}$.
ρ	The smallest number such that every row of A has at most ρ number-changes.
S	Segment of A. For $m = 1$, $S[1..j]$ is a vector.
$v(S)$	Value of a segment of A.
$\mathcal{S}$	Set of segments that are a segmentation for $A[1..j]$.
Signature of $\mathcal{S}$	Multi-set of all values of segments in $\mathcal{S}$ ending in index j.
ϕ	Partition of a value in A; in our context, the partition of $A[j]$.
$\Phi_{j-1}(\varphi)$	Set of partitions of $A[j-1]$ that can be obtained from φ by deleting at most one element and adding at most one element.
$f(j, \phi)$	Minimum number of segments in an almost-compact segmentation $\mathcal{S}$ of $A[1..j]$ that has signature ϕ.
$p(H)$	Number of partitions of the integer H.
$m_t(\mathcal{M})$	Multiplicity of element t in a multi-set $\mathcal{M}$.
$\mathcal{M}$	Multi-set with entries from the universe $[H]$ can be described via the H-tuple $(m_1(\mathcal{M}), \ldots, m_H(\mathcal{M}))$.
$\mathcal{M}(\mathcal{S})$	Multi-set defined by values of segments in segmentation $\mathcal{S}$.
$c(A[i])$	Complexity of row i of A.
$f'(j, \phi, \nu)$	For partition ϕ of $A[j]$ and a multiset ν over $[H]$, this function equals 1 if there $f'(j, \phi, \nu)$ exists a segmentation $\mathcal{S}$ of $A[1..j]$ with signature ϕ and multi-set $\mathcal{M}(\mathcal{S}) \subseteq \nu$; 0 otherwise.
$f''(j, \phi, \nu)$	For partition ϕ of $A[j]$ and a multiset ν over $[H]$, this function equals the minimum possible number of 1-segments in a segmentation $\mathcal{S}$ of $A[1..j]$ with signature ϕ and multi-set $\mathcal{M}(\mathcal{S}) \subseteq \nu \cup (\infty, 0, \ldots, 0)$.
$\mathbb{M}$, $\mathbb{M}_{lex}$	Set of interesting multi-sets for the full-matrix and lex-min problem.

Table 1. Summary of frequently used notation.

element $t \in [H]$ from multi-set $\mathcal{M}$ means increasing (resp. decreasing) $m_t(\mathcal{M})$ by one (while keeping $m_t(\mathcal{M}) \geq 0$). Finally, we say that $\mathcal{M}_1$ *is contained* in $\mathcal{M}_2$ and write $\mathcal{M}_1 \subseteq \mathcal{M}_2$ whenever $m_t(\mathcal{M}_1) \leq m_t(\mathcal{M}_2)$ for $t \in [H]$.

In order to help the reader, we aggregate our frequently used notation in Table 1.

The key idea of our algorithm is to compute the best almost-compact segmentation of $A[1..j]$ subject to a given signature. Thus define a function f as follows:

Given an integer j and a partition ϕ of $A[j]$, let $f(j, \phi)$ be the minimum number of segments in an almost-compact segmentation $\mathcal{S}$ of $A[1..j]$ that has signature ϕ.

We will show that $f(j, \phi)$ can be computed recursively. To simplify computation we will use $f(0, \cdot)$ as a base case; we assume that $A[0] = A[n+1] = 0$. The only possible partition of 0 is the empty partition, and so $f(0, \emptyset) = 0$ is our base case.

Given a partition ϕ of $A[j]$, let $\Phi_{j-1}(\phi)$ be the set of those partitions of $A[j-1]$ that can be obtained from ϕ by deleting at most one element, and then adding at most one element. The recursive formula for f will be given in Lemmas 2 and 3.

Lemma 2. *For $j \geq 1$, $f(j, \phi) \geq \min_{\psi \in \Phi_{j-1}(\phi)} \{f(j-1, \psi) + ||\phi - \psi||\}$*

Proof. Consider an almost-compact segmentation $\mathcal{S}_j$ of $A[1..j]$ that achieves the left-hand side, i.e., its signature is ϕ and $|\mathcal{S}_j| = f(j, \phi)$. We have four kinds of segments in $\mathcal{S}_j$: (1) Those that end at index $j-2$ or earlier, (2) those that end at $j-1$ (there can be at most one, since $\mathcal{S}_j$ is almost-compact), (3) those that end at j and start at $j-1$ or earlier, and (4) those that end at j and begin at j (there can be at most one).

Let $\mathcal{S}_{j-1}$ be the segmentation of $A[1..j-1]$ obtained from $\mathcal{S}_j$ by taking all segments of type (1)–(3), and deleting the last entry (at index j). The value of each segment in $\mathcal{S}_{j-1}$ is defined as the value of the corresponding segment in $\mathcal{S}_j$. Note that $\mathcal{S}_{j-1}$ is also almost-compact. The signature ψ of $\mathcal{S}_{j-1}$ is the same as ϕ, except that the value of the (unique) segment of type (4) (if any) has been removed, and the value of the (unique) segment of type (2) (if any) has been added. So $\psi \in \Phi_{j-1}(\phi)$.

If both a segment of type (4) and a segment of type (2) exist in $\mathcal{S}_j$, then they necessarily have different non-zero values (otherwise they could be combined, contradicting the minimality of $\mathcal{S}_j$). Hence $||\phi - \psi||$ is exactly the number of segments of type (4). So $|\mathcal{S}_{j-1}| = |\mathcal{S}_j| - ||\phi - \psi||$, which proves the claim. □

Lemma 3. *For $j \geq 1$, $f(j, \phi) \leq \min_{\psi \in \Phi_{j-1}(\phi)} \{f(j-1, \psi) + ||\phi - \psi||\}$.*

Proof. Let $\psi \in \Phi_{j-1}(\phi)$ be a partition of $A[j]$ that achieves the minimum on the right-hand side. Let $\mathcal{S}_{j-1}$ be an almost-compact segmentation that achieves $f(j-1, \psi)$, i.e., it is a segmentation of $A[1..j-1]$ with signature ψ and cardinality $f(j-1, \psi)$.

Define a segmentation $\mathcal{S}_j$ of $A[1..j]$ as follows. Each segment of $\mathcal{S}_{j-1}$ that ends before index $j-1$ is extended by setting its jth entry to be 0 and added to $\mathcal{S}_j$. For each value t in $\psi - \phi$, there must be an t-segment in $\mathcal{S}_{j-1}$ that ends at index $j-1$; add this segment to $\mathcal{S}_j$ and let it end at $j-1$ (i.e., set its jth entry to 0). For each value t in $\psi \cap \phi$, there must be an t-segment in $\mathcal{S}_{j-1}$ that ends at index $j-1$; add this segment to $\mathcal{S}_j$ and extend it to j (i.e., set its jth entry to 1). In all the preceding cases, the value of each segment in $\mathcal{S}_j$ is defined as the value of the corresponding segment in $\mathcal{S}_{j-1}$.

Finally, for each value t in $\phi - \psi$, define a new segment in $\mathcal{S}_j$ that starts at j and has value t. One easily verifies that $\mathcal{S}_j$ has signature ϕ, and therefore it is a segmentation of $A[1..j]$, since ϕ is a partition of $A[j]$. We can convert it to an almost-compact segmentation as in the proof of Lemma 1. Also, $|\mathcal{S}_j| = |\mathcal{S}_{j-1}| + ||\phi - \psi||$, which proves the result. □

Theorem 1. *The single-row segmentation problem can be solved in $O(p(H)\,H \cdot n)$ time and $O(n + p(H)H)$ space, where $p(H)$ is the number of partitions of H.*

Proof. It follows from Lemmas 2 and 3 that, for $j \geq 1$, $f(j, \phi) = \min_{\psi \in \Phi_{j-1}(\phi)}\{f(j-1, \psi) + ||\phi - \psi||\}$. Here, the idea is to evaluate $f(j, \phi)$ recursively with a dynamic programming approach; the optimal value can then be found in $f(n+1, \emptyset)$. To achieve the time complexity, we need to store the partitions in a suitable data structure. The key property here is that any partition ϕ of $A[j] \leq H$ has $O(\sqrt{H})$ distinct integers in the set $\{1, \ldots, H\}$. Thus, we can describe a partition in $O(\sqrt{H})$ space, and store it (using a trie[5]) so that it can be located in $O(\sqrt{H})$ time. We give the details of this data structure in the appendix. Using such tries, we can also generate all relevant partitions efficiently; since $|\Phi_{j-1}(\phi)| \in O(\sqrt{H})$, we can therefore compute $f(j, \phi)$ from the stored values of $f(j-1, \cdot)$ in $O(H)$. Summing over all ϕ and $j = 1, \ldots, n$ then gives the desired time bound. □

Note that the algorithm is fixed-parameter tractable with respect to parameter H. It is known that $p(H) \leq e^{\pi \cdot \sqrt{\frac{2 \cdot H}{3}}}$ [12], so this algorithm is in fact polynomial as long as $H \in O(\log^2 n)$. In the present form, it only returns the size of the smallest segmentation, but standard dynamic programming techniques can be used to retrieve the segmentation in the same running time with an $O(\log n)$ space overhead. Finally, we note that the space requirement can be improved by a factor of H at the expense of an additional $\log H$ factor in the running time (see Section B in the appendix).

3 Full-matrix segmentation

In this section, we give an algorithm that computes the optimal segmentation for a full matrix, and which is polynomial as long as H is a constant.

3.1 Segmenting a row under constraints

The difficulty of full-matrix segmentation lies in that rows cannot be solved independently of each other, since an optimal segmentation of a full matrix does not mean that the induced segmentations of the rows are optimal. Consider for example

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$$

[5] There are much simpler data structures, e.g. we could store partitions as entries in a H-dimensional array, but this would use more space and/or be slower.

which is an optimal segmentation, but the induced segmentation for the third row is not optimal.

If $\mathcal{S}$ is a segmentation, then let $m_t(\mathcal{S})$ be the number of t-segments in $\mathcal{S}$; note that this defines a multi-set over $[H]$ which we refer to as the *multi-set $\mathcal{M}(\mathcal{S})$ defined by segmentation $\mathcal{S}$*.

We now want to compute whether a row $A[1..n]$ has a segmentation $\mathcal{S}$ such that $\mathcal{M}(\mathcal{S}) \subseteq \nu$ for some given multi-set ν. We do this again with dynamic programming, by further restricting the segmentation to the first j elements of the row of the matrix, and by restricting the signature (the implied partition of the last entry). Thus define the following:

> *Given an integer j, a partition ϕ of $A[j]$, and a multiset ν over $[H]$, define $f'(j, \phi, \nu)$ to be 1 if there exists a segmentation $\mathcal{S}$ of $A[1..j]$ with signature ϕ and multi-set $\mathcal{M}(\mathcal{S}) \subseteq \nu$. Define $f(j, \phi, \nu)$ to be 0 otherwise.*

For example, consider $A = [1\ 3\ 2\ 4]$, $\phi = \{1, 3\}$ and $\nu = \{1, 1, 1, 2, 3\}$. Then $f'(4, \phi, \nu)$ asks whether we can segment A such that at index 4 we use one 1-segment and one 3-segment, and overall we use at most three 1-segments, at most one 2-segment, and at most one 3-segment. The answer in this case is yes ($[1\ 3\ 2\ 4] = [1\ 1\ 0\ 0] + [0\ 2\ 2\ 0] + [0\ 0\ 0\ 1] + [0\ 0\ 0\ 3]$), so $f'(4, \phi, \nu) = 1$. Note that we were allowed one more 1-segment than was actually used; this is acceptable since the multi-set of the segmentation is allowed to be a subset of ν.

We claim that $f'(\cdot, \cdot, \cdot)$ has a simple recursive formula. The base case is again $j = 0$ (as before we assume $A[0] = A[n+1] = 0$). Since $A[0] = 0$, the only possible signature can be $\emptyset$, and we have $f'(0, \emptyset, \nu) = 1$ for all possible multi-sets ν. For $j \geq 1$, we can compute $f'(j, \phi, \nu)$ from $f'(j-1, \cdot, \cdot)$ as follows:

Lemma 4. *For all $j \geq 1$,*

$$f'(j, \phi, \nu) = \max_{\psi \text{ is a partition of } A[j-1]} f'(j-1, \psi, \nu - (\phi - \psi)). \tag{1}$$

Before proving this, we will illustrate it with the above example of $A = [1\ 3\ 2\ 4]$, $\phi = \{1, 3\}$ and $\nu = \{1, 1, 1, 2, 3\}$. Let $\psi = \{2\}$ and $\nu' = \{1, 1, 2\}$. Then $f'(3, \psi, \nu') = 1$ since $[1\ 3\ 2] = [1\ 1\ 0] + [0\ 2\ 2]$. Furthermore, we have $\phi - \psi = \{1, 3\}$ and $\nu - (\phi - \psi) = \{1, 1, 2\} = \nu'$. Therefore, the formula says that $f'(4, \phi, \nu)$ should be 1, which indeed it is.

The following proof is quite similar in spirit to the proofs of Lemmas 2 and 3, except that we also need to keep track of how the multi-sets of segmentations change.

Proof (of Lemma 4). Assume first that $f'(j, \phi, \nu) = 1$, so we have a segmentation $\mathcal{S}_j$ of $A[1..j]$ with signature ϕ and $\mathcal{M}(\mathcal{S}) \subseteq \nu$. Let $\mathcal{S}_{j-1}$ be its implied segmentation of $A[1..j-1]$, and let it have signature ψ. Then $\phi - \psi$ contains exactly those values of segments of $\mathcal{S}_j$ that begin (and end) at j, and hence are not in $\mathcal{S}_{j-1}$. Since $\mathcal{S}_j$ contains at most $m_t(\nu)$ segments of value t, therefore $\mathcal{S}_{j-1}$ contains at most $m_t(\nu) - m_t(\phi - \psi)$ segments of value t. Thus segmentation $\mathcal{S}_{j-1}$ has signature ψ and satisfies $\mathcal{M}(\mathcal{S}_{j-1}) \subseteq \nu - (\psi - \phi)$, which proves $f'(j-1, \psi, \nu - (\psi - \phi)) = 1$ and hence the right-hand side is 1.

Now assume that the right-hand side is 1, say for partition ψ there exists a segmentation $\mathcal{S}_{j-1}$ that has signature ψ and satisfies $\mathcal{M}(\mathcal{S}_{j-1}) \subseteq \nu - (\phi - \psi)$. By adding a new t-segment (beginning and ending at j) for each value t in $\phi - \psi$, and extending a segment to also cover index j for each value t in $\psi \cap \phi$, we can create a segmentation $\mathcal{S}_j$ of $A[1..j]$ that has signature ϕ and $\mathcal{M}(\mathcal{S}_j) \subseteq \nu$, so $f'(j, \phi, \nu) = 1$ as desired. □

We now turn to the run-time of actually computing f'. In the above definition, we have not imposed any bounds on ν, other than that it is a multi-set over $[H]$. But clearly we can restrict the multi-sets considered. Assume for a moment that we know an optimal segmentation $\mathcal{S}^*$ of the full matrix. We call a multi-set ν *relevant* if $\nu \subseteq \mathcal{M}(\mathcal{S}^*)$. Clearly it suffices to compute f' for all relevant multi-sets.

To find (a superset of) relevant multi-sets without knowing $\mathcal{S}^*$, we exploit that $\mathcal{M}(\mathcal{S}^*)$ cannot contain too many segments of the same value. Let $\Delta[i][j] := A[i][j] - A[i][j-1]$ for $j \in [n+1]$, where we assume that $A[i][0] = A[i][n+1] = 0$. We say that there is a *marker* between index $j-1$ and j in row i if $\Delta[i][idxn] \neq 0$, i.e., if the value in A changes. Let ρ_i be the number of markers in row i, and let $\rho = \max_i \rho_i$.

From now one, whenever we consider a segmentation of a row, we will assume that it has been *standardized* in the following way: (1) Every segment s begins and ends at a marker. For if it doesn't, then some other segments(s) must end where s begins (or vice versa), and by moving all these end-points to the nearest marker, we retain a segmentation without adding any new segments (and perhaps even deleting some.) (2) Whenever a segment ends at a marker, then there is no other segment of the same value that begins at that marker. For otherwise the two segments could be combined into one.

We now prove a useful bound on segmentations of the full matrix A.

Lemma 5. *If all rows of A have at most ρ markers, then there exists a minimum cardinality segmentation that has at most $\rho/2$ t-segments for all $t \in [H]$.*

Proof. Among all optimal segmentations of A, let $\mathcal{S}^*$ be the one that maximizes $\sum_{t=1}^{H} m_t(\mathcal{M}(\mathcal{S}^*))^2$. Assume for contradiction that there is some $t \in [H]$ for which $\mathcal{S}^*$ contains $r > \rho/2$ segments of value t. We will then create another segmentation $\mathcal{S}'$ that uses $r-1$ segments of value t, and one additional $(2t)$-segment. Hence $\mathcal{S}'$ has the same cardinality as $\mathcal{S}^*$, but at the same time $\sum_{t=1}^{H} m_t(\mathcal{M}(\mathcal{S}'))^2 > \sum_{t=1}^{H} m_t(\mathcal{M}(\mathcal{S}^*))^2$, contradicting the choice of $\mathcal{S}^*$.

To see why such an $\mathcal{S}'$ exists, consider any row of A and its segmentation by $\mathcal{S}^*$. As explained above, we assume that this segmentation is standardized. If the segmentation uses at most $r-1$ segments of value t, then we use the exact same segmentation in $\mathcal{S}'$. If it uses $r > \rho/2$ segments of value t, then there must be in this segmentation two t-segments that both begin at the same marker, or both end at the same marker, say the former. We can replace these two segments by a $2t$-segment, followed by one (possibly empty) t-segment once the first of the two segments ends. Hence again we obtain a segmentation of the row that can be used for $\mathcal{S}'$. Therefore, in every row we can change the segmentation by $\mathcal{S}^*$ into one that can be used for $\mathcal{S}'$, proving that $\mathcal{S}'$ exists, a contradiction. □

Now let $\mathbb{M}$ be all those multi-sets over $[H]$ where all multiplicities are at most $\rho/2$; this contains all relevant multi-sets. We store these in an H-dimensional array with indices in $[0..\rho/2]$; this takes $O((\rho/2)^H)$ space, and allows lookup of a multi-set in $O(H)$ time. Using (1), one can easily see that we can compute the values $f'(j, \phi, \nu)$ as shown in Algorithm 1.

Algorithm 1

```
1: Let $\mathbb{M}$ be all multi-sets where all multiplicities are at most $\rho/2$.
2: for all multi-sets $\nu$ in $\mathbb{M}$ do
3:   Initialize $f'(0, \emptyset, \nu) = 1$.
4:   for $j = 1, \dots, n+1$ do
5:     for all multi-sets $\nu$ in $\mathbb{M}$ do
6:       for all partitions $\phi$ of $A[j]$ do
7:         Initialize $f'(j, \phi, \nu) = 0$
8:         for all partitions $\psi$ of $A[j-1]$ do
9:           Compute $\nu' = \nu - (\phi - \psi)$
10:          if $f'(j-1, \psi, \nu') = 1$ then
11:            Set $f'(j, \phi, \nu) = 1$ and break
12:          end if
13:        end for
14:      end for
15:    end for
16:  end for
17: end for
```

The running time of Algorithm 1 is determined by the time to compute ν' and looking up $f'(j-1, \psi, \nu')$. Computing ν' (given ν, ϕ and ψ) can certainly be done in $O(H)$ time. To look up $f'(j-1, \psi, \nu')$, we first look up ν in the appropriate data structure in $O(H)$ time. With each multi-set $\nu \in \mathbb{M}$, we store all partitions of $A[j-1]$ and of $A[j]$ (for the current value of j), and with each of them, the values of $f'(j-1, \psi, \nu)$ and $f'(j, \psi, \nu)$, respectively. Looking up or changing these values (given ν and ψ) can then be done in $O(\sqrt{H})$ time by storing partitions in tries.

So lines 9-11 require $O(H)$ time. They are executed $p(H)$ times from line 8, $p(H)$ times from line 6, $|\mathbb{M}|$ times from line 5, and $n+1$ times from line 4; this is $O(n(\rho/2+1)^H p(H)^2 H)$ time.

As for the space requirements, we need to store all relevant multi-sets, and with each, all partitions of $A[j-1]$ and $A[j]$, which takes $O(H)$ space per partition. So the total space is $O(p(H)H(\rho/2)^H)$.

Lemma 6. *Consider one row $A[1..n]$. In $O(n(\rho/2)^H p(H)^2 H)$ time and $O(p(H)H(\rho/2)^H)$ space we can compute an H-dimensional binary array $\mathcal{F}$ such that for any $m_1, \ldots, m_H \leq \rho/2$ we have $\mathcal{F}(m_1, \ldots, m_H) = 1$ if and only if there exists a segmentation of $A[1..n]$ that uses at most m_t segments of value t for $t \in [H]$.*

3.2 Full-matrix

Solving the full-matrix problem is now quite simple. For all rows i, compute the table $\mathcal{F}_i$ described in Lemma 6. This takes time $O(mn(\rho/2)^H p(H)^2 H)$ total. The space is $O(p(H)H(\rho/2)^H)$ per row, but once done with a row i we only need to keep the $O((\rho/2)^H)$ values for the corresponding table $\mathcal{F}_i$; therefore, in total, it is $O(\max\{m, p(H)H\}(\rho/2)^H)$. Now, in $O(m(\rho/2)^H)$ time find the numbers $m_1, \ldots, m_H$ for which $\mathcal{F}_i(m_1, \ldots, m_H)$ is 1 for *all* rows i and for which $m_1 + \cdots + m_H$ is minimized.

Claim. The values $m_1, \ldots, m_H$ exist, and the optimal segmentation has size $m_1 + \cdots + m_H$.

Proof. Let $\mathcal{S}^*$ be an optimal segmentation of the full matrix with at most $\rho/2$ segments of value t for all $t \in [H]$. By Lemma 5, such a segmentation exists. Let m_t^* be the number of t-segments in $\mathcal{S}^*$. The row segmentations induced by $\mathcal{S}^*$ have at most m_t^* segments of value t for all $t \in [H]$, so $\mathcal{F}_i(m_1^*, \ldots, m_H^*)$ is 1 for all rows. Thus $m_1, \ldots, m_H$ exist and by their choice as minimizing the sum we have $m_1 + \cdots + m_H \leq m_1^* + \cdots + m_B^* = |\mathcal{S}^*|$.

For the other inequality, by choice of the m_t's, every row i has a segmentation $\mathcal{S}_i$ with at most m_t segments of value t. We can combine these segmentations with a greedy packing algorithm (see also [6]): Define m_t matrix-segments of value t by taking from each row a row-segment of value t if there is one left. Repeat for all t. This gives a segmentation $\mathcal{S}$ of the full matrix of size $m_1 + \cdots + m_H$, so $|\mathcal{S}^*| \leq m_1 + \cdots + m_H$.

Combining the two inequalities shows $|\mathcal{S}^*| = m_1 + \cdots + m_H$. □

This implies the next result.

Theorem 2. *The full-matrix segmentation problem can be solved in $O(mn(\rho/2)^H p(H)^2 H)$ time and $O(\max\{m, p(H)H\}(\rho/2)^H)$ space if each row has at most ρ markers.*

Note that one could view our result as a fixed-parameter tractability result, where the parameter is $H + \rho$. However, normally ρ will be large. In particular, if a natural pre-processing step is applied that removes from each row of A any consecutive identical numbers (this does not affect the size of the optimum solution), then $\rho = n + 1$. We therefore prefer to re-phrase our theorem to express the worst-case run-time in terms of m, n and H only. Note that $\rho \leq n + 1$ always, so the run-time becomes $O(mn^{H+1}p(H)^2 H/2^H)$. Recall that $p(H) \leq e^{\pi\sqrt{\frac{2H}{3}}} \leq e^{2.6\sqrt{H}}$ and, therefore, $Hp(H)^2 \leq He^{5.2\sqrt{H}} = 2^{\lg(H) + 5.2\sqrt{H}\lg(e)} \leq 2^{8.6\sqrt{H}}$, implying that $p(H)^2 H/2^H \in O(2^{-(1-\epsilon)H})$ for some arbitrarily $\epsilon > 0$ for H sufficiently large.

Corollary 1. *The full-matrix segmentation problem can be solved in $O(mn^{H+1}/2^{(1-\epsilon)H})$ time, where $\epsilon > 0$ is an arbitrarily small constant, and $O(mn^H)$ space.*

3.3 Further improvements of the complexity

We sketch a further improvement that removes a factor of n from the running time. Recall that the function $f'(j, \phi, \nu)$ was defined to be 1 if and only if there exists a segmentation $\mathcal{S}$ of $A[1..j]$ with signature ϕ and multi-set $\mathcal{M}(\mathcal{S}) \subseteq \nu$. In its place, we can instead define a function $f''(j, \phi, \nu)$, which contains the minimum number of 1-segments in a segmentation $\mathcal{S}$ of $A[1..j]$ with signature ϕ and multi-set $\mathcal{M}(\mathcal{S}) \subseteq \nu + \nu_1$. Here, ν_1 is the multi-set that has $m_1(\nu_1) = \infty$ and $m_t(\mathcal{M}_1) = 0$ for all $t \neq 1$. In other words, the segmentation that defines f'' is restricted in the number of t-segments only for $t > 1$, and the restriction on 1-segments is expressed in the return-value of f''. In particular, the value of $f''(j, \phi, \nu)$ is independent of the first multiplicity of ν, and hence must be computed only for those ν with $m_1(\nu) = 0$; there are only $(\rho/2 + 1)^{H-1}$ such multi-sets ν.

It remains to argue that f'' can be computed efficiently, with a similar formula as for f'. This is quite simple. To compute $f''(j, \phi, \nu)$, try all possible partitions ψ of $A[j-1]$, compute $\nu' = \nu - (\phi - \psi)$, and let ν'' be ν' with its first multiplicity changed to 0. Look up the value $f''(j-1, \psi, \nu'')$ and add to it the number of 1s in $\phi - \psi$. This gives one possible candidate for a segmentation; we find the best one by minimizing over all ψ. We leave the formal proof of correctness to the reader.

We can hence compute $f''(n+1, \emptyset, \nu)$ for all $(\rho/2)^{H-1}$ multi-sets ν in $O(n(\rho/2)^{H-1}\ p(H)^2 H)$ time. Doing this for all rows, we can compute the maximum of the values $f''(n+1, \emptyset, \nu)$ over all rows. The optimum segmentation can then be found by choosing the one that minimizes this maximum plus $||\nu||$ over all ν. As before, this only adds an extra $O(m)$ factor to the run-time, which is hence $O(mn(\rho/2)^{H-1} p(H)^2 H)$, and similarly as before this can be simplified to $O(mn^H/2^{(1-\varepsilon)H})$.

Theorem 3. *The full-matrix segmentation problem can be solved in $O(mn^H/2^{(1-\epsilon)H})$ time, for $\epsilon > 0$ an arbitrarily small constant, and $O(mn^{H-1})$ space.*

3.4 Solving the lex-min problem

Recall that the lex-min problem is that of finding a minimum cardinality segmentation among those with minimum *beam-on-time*, defined as the total value $\sum_{S \in \mathcal{S}} v(S)$ of the segmentation. Here, we show how to apply our techniques to achieve a speed up in solving this problem. To this end, we need the notion of the *complexity of row* $A[i]$ which is defined as:

$$c(A[i]) := \frac{1}{2}\sum_{j=1}^{n+1} |\Delta[i][j]| = \sum_{j=1}^{n+1} \max\{0, \Delta[i][j]\} = \sum_{j=1}^{n+1} - \min\{0, \Delta[i][j]\},$$

where as before $\Delta[i][j] := A[i][j] - A[i][j-1]$ for $j \in [n+1]$.

Importantly, is was shown in [14] that the minimum beam-on time can be computed efficiently; it is $c(A) := \max_i\{c(A[i])\}$. To solve the lex-min problem, we simply have to change our focus regarding the set $\mathbb{M}$ of interesting multi-sets. Instead of the relevant multi-sets as used earlier, where each multiplicity is at most $\rho/2$, we need all multi-sets ν such that $\sum_{t=1}^{H} t \cdot m_t(\nu)$ equals the minimum beam-on time. Let $\mathbb{M}_{lex}$ be the set of these multi-sets and their subsets. While Lemma 5 no longer applies, we still obtain a useful bound on the size $\mathbb{M}_{lex}$.

Lemma 7. *If all rows of A have at most ρ markers, then there exists a minimum cardinality segmentation among all those that have minimum beam-on time that has at most $\rho - 1$ t-segments for all $t \in [H]$. Moreover, for $t > H/2$, there are at most $\rho/2$ t-segments.*

Proof. The proof is almost identical to the one of Lemma 5, except that we need to choose $\mathcal{S}'$ more carefully to ensure that it, too, has minimum beam-on time. So let $\mathcal{S}^*$ be a minimum cardinality segmentation among all those that have minimum beam-on time. As usual, we assume that $\mathcal{S}^*$ has been standardized; this does not affect the beam-on time.

It is then near-trivial that $\mathcal{S}^*$ has at most $\rho/2$ t-segments for $t > H/2$: No two such segments can overlap (since their combined value exceeds H), and so no two of them can share a marker (since $\mathcal{S}^*$ is standardized); since each touches two markers therefore there can be at most $\rho/2$ of them.

The other claim is more complicated. Assume for contradiction that there is some $t \in [H]$ for which $\mathcal{S}^*$ contains $r \geq \rho$ segments of value t. We will then create another segmentation $\mathcal{S}'$ that uses $r-2$ segments of value t, and one additional $2t$-segment. Hence $\mathcal{S}'$ has the same beam-on time and a smaller cardinality as $\mathcal{S}^*$, contradicting the choice of $\mathcal{S}^*$.

To see why such an $\mathcal{S}'$ exists, consider any row of A and its segmentation by $\mathcal{S}^*$. If the segmentation uses at most $r-2$ segments of value t, then we use the exact same segmentation in $\mathcal{S}'$. If it uses $r-1 \geq \rho - 1 \geq \rho/2$ segments of value t, then as in the proof of Lemma 5, two of them must start at the same marker, and we remove them and replace them by a $(2t)$-segment. The resulting segmentation has less than $r-2$ segments of value t and one of value $2t$ and so can be used for $\mathcal{S}'$.

Now finally assume that some row contains $r \geq \rho$ segments of value t. Define an auxiliary graph H as follows: H has a vertex for every marker, and an edge between two markers if and only if there exists a t-segment that begins at one of them and ends at the other. H has ρ vertices and $r \geq \rho$ edges, and hence contains a cycle C. Let $s_1 = [j_1, j_2]$ be a shortest segment on C (as measured by $j_2 - j_1$), and let s_0 and s_2 be its neighbours on C.

Rename, if needed, so that s_0 and s_2 share the marker at j_1 and j_2, respectively, with s_1. Since $\mathcal{S}^*$ is standardized, we have $s_0 = [j_1, j_3]$ and $s_2 = [j_0, j_2]$. Since s_1 was the shortest segment on C, we have $j_0 \leq j_1 \leq j_2 \leq j_3$. Now define a segment $[j_1, j_2]$ of value $2t$, and if $s_0 \neq s_2$ another segment $[j_0, j_3]$ of value t, and use these segments instead of s_0, s_1, s_2. One easily verifies that this is also a segmentation of the row, and it can be used for $\mathcal{S}'$. This proves that $\mathcal{S}'$ exists, a contradiction. $\square$

We can hence find and store a (super-set of) $\mathbb{M}_{lex}$ by using all entries in an H-dimensional array $[0, \rho]^{\lfloor H/2 \rfloor} \times [0, \rho/2]^{\lceil H/2 \rceil}$, and there are $O(\rho^H / 2^{H/2})$ such multi-sets. We will compute $f''(n+1, \emptyset, \nu)$ for all such multi-sets ν, and then pick a multi-set ν for which $||\nu|| + \sum_{t=1}^{H} m_t(\nu)$ is minimized, and for which $\sum_{t=1}^{H} t m_t(\nu)$ equals $c(A)$. This is then the multi-set used for a minimum segmentation among those with minimum beam-on time; we can find the actual segmentation by re-tracing the computation of $f''(n+1, \emptyset, \nu)$.

By the same analysis used for the minimum cardinality segmentation problem, and the improvement described in the previous Section 3.3, we have:

Theorem 4. *The lex-min problem can be solved in $mn^H / 2^{(\frac{1}{2}-\varepsilon)H}$ time and with $O(mn^{H-1})$ space.*

Recall that Kalinowski's algorithm in [15] has a time complexity of $O(2^H \sqrt{H} \cdot m \cdot n^{2H+2})$. So we obtain an almost-quadratic improvement in the time complexity. Finally, we note that it is intuitively reasonable that our algorithm can be applied to the lex-min problem since the restriction on the space of feasible solutions that the beam-on time be minimized can be captured by modifying appropriately the set of interesting multi-sets $\mathbb{M}_{lex}$.

4 The special case of $H = 2$

For $H = 2$ (i.e., a 0/1/2-matrix), the algorithm of Section 3.3 has run-time $O(mn^2)$. As we show in this section, however, yet another factor of n can be shaved off by analyzing the structure of the rows more carefully. In a nutshell, the function f'' of Section 3.3 can be computed from the structure of the row alone, without needing to go through all possible signatures; we explain this now. Throughout Section 4, we assume that all entries in the intensity matrix are 0, 1, or 2.

4.1 Single row for $H = 2$

As before, let $A[1..n]$ be a single row of the matrix. Consider a maximal interval $[j', j'']$ such that $A[j'..j'']$ has all its entries equal to 2. We call $A[j'..j'']$ a *tower* if $A[j'-1]$ and $A[j''+1]$ both equal 0, a *simple step* if one of $A[j'-1]$ and $A[j''+1]$ equals 1 and the other 0, and a *double step* otherwise. (As usual we assume that $A[0] = A[n+1] = 0$.) We use t, s and u to denote the number of towers, simple steps and double steps, respectively. Figure 2 illustrates how interpreting $A[i] = t$ as t blocks atop each other gives rise to these descriptive names.

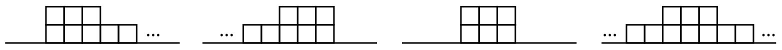


Fig. 2. Two kinds of simple steps, a tower, and a double-step.

Recall that $c(A[i]) = \sum_{j=1}^{n+1} \max\{\Delta[i][j], 0\}$ is the complexity of a row i of a full matrix A; we use $c(A)$ for the complexity of the single row A under consideration.

Lemma 8. *Define $g(d)$ as follows:*

$$g(d) := \begin{cases} c(A) - 2d & \text{if } d < t, \\ c(A) - t - d & \text{if } t \leq d \leq s + t, \\ c(A) - 2t - s & \text{if } t + s < d. \end{cases}$$

Then for any $d \geq 0$, $f''(n+1, \emptyset, \{0, d\}) = g(d)$. In other words, any segmentation $\mathcal{S}$ of A with at most d segments of value 2 has at least $g(d)$ segments of value 1. Moreover, there exists a segmentation that has at most d segments of value 2 and at most $g(d)$ segments of value 1.

Proof. Let $\mathcal{S}$ be a segmentation of A that uses at most d segments of value 2. As before, we assume that $\mathcal{S}$ has been standardized, which can be done without increasing the number of 2-segments. Therefore, any tower, step or double-step of A is either entirely covered by a 2-segment, or it does not intersect any 2-segment.

Let s_2, t_2 and u_2 be the number of steps, towers, and double-steps (respectively) that are entirely covered by a 2-segment. We claim the the number of 1-segments of $\mathcal{S}$ is $c(A) - s_2 - 2t_2$, and can prove this by induction on $s_2 + t_2 + u_2$. If $s_2 + t_2 + u_2 = 0$, then $\mathcal{S}$ has only 1-segments, and since $\mathcal{S}$ is standardized, the number of 1-segments equals $c(A)$. If, say, $t_2 > 0$, then let A' be the vector obtained from A by removing a tower that is covered by a 2-segment (i.e., by replacing the 2s of that tower by 0s), and let $\mathcal{S}'$ be the segmentation of A' obtained from $\mathcal{S}$ by removing the 2-segment that covers that tower. Then A' has $t_2' = t_2 - 1$ towers covered by 2-segments, and furthermore $c(A') = c(A) - 2$. Since $\mathcal{S}$ and $\mathcal{S}'$ have the same number of 1-segments, the claim easily follows by induction. Similarly one proves the claim by induction if $s_2 > 0$ or $u_2 > 0$.

Therefore the number of 1-segments in $\mathcal{S}$ is $c(A) - s_2 - 2t_2$. We also know that $s_2 + t_2 + u_2 \leq d$. So to get a lower bound on the number of 1-segments, we should minimize $c(A) - s_2 - 2t_2$, subject to $s_2 + t_2 + u_2 \leq d$ and the obvious $0 \leq s_2 \leq s$, $0 \leq t_2 \leq t$ and $0 \leq u_2 \leq u$. The bound now easily follows by distinguishing whether $d < t$ (the minimum is at $t_2 = d$, $s_2 = u_2 = 0$), or $t \leq d < t + s$ (minimum at $t_2 = t$, $s_2 = d - t$, $u_2 = 0$) or $t + s < d$ (minimum at $t_2 = t$, $s_2 = s$, $u_2 = 0$.)

For the second claim, we obtain such a segmentation by using $\min\{d, t\}$ 2-segments for towers, then $\min\{d - t, s\}$ 2-segments for stairs if $d \geq t$, and cover everything else by 1-segments. □

The crucial idea for $H = 2$ is that since $g(\cdot)$ can be described explicitly with only three linear equations that can easily be computed, we can save space and time by not storing $f''(n+1, \emptyset, \{0, d\})$ explicitly as an array of length $\rho/2 + 1$, and not spending $O(n \cdot \rho/2)$ time to fill it.

4.2 Full matrix segmentation for $H = 2$

As in Section 3.3, to solve the full-matrix problem we need to find the value d^* that minimizes $d + \max_i\{f_i''(n+1, \emptyset, \{0, d\})\} =: D$, where $f_i''(\cdot)$ is function $f''(\cdot) = g(\cdot)$ for row i. We can hence find the optimal segmentation of A as follows. Compute the complexity and the number of towers and stairs in each row; this takes $O(mn)$ time total. Each $f_i''(\cdot)$ is then the maximum of three lines defined by these numbers. Hence $d + \max_i\{f_i''(n+1, \emptyset, \{0, d\})\}$ is the maximum of $3m$ lines. We hence can compute D (and with it d^*) by taking the intersection of the upper half-spaces defined by the $3m$ lines (this can be done in $O(m)$ expected time easily, and in $O(m)$ worst-case time with a complicated algorithm [13]), and then finding the grid point with the smallest y-coordinate in it.

Once we found d^*, we can easily compute a segmentation of each row that has at most $D - d^*$ segments of value 1 and at most d^* segments of value 2 (see the proof of Lemma 8) and combine them into a segmentation of the full matrix with the greedy-algorithm; this can all be done in $O(mn)$ time. Thus the overall run-time is $O(mn)$.

Theorem 5. *A minimum cardinality segmentation of an intensity matrix with values in $\{0, 1, 2\}$ can be found in $O(mn)$ time.*

An immediate application of this result is that it can be combined with the $O(\log h)$ approximation algorithm in [6]. While approximation guarantee remains unchanged, this should result in improved solutions in practice while not substantially increasing the running time.

One naturally asks whether this approach could be extended to higher values of H. The main obstacles to doing this is the function $g(d)$. The overall approach would work well for $H = 3$ if we had a function $g_i(d_2, d_3)$ for each row $A[i]$ lower bounding the number of 1-segments in a segmentation if it has at most d_2 segments of value 2 and d_3 segments of value 3. It seems likely that the functions $g_i(d_2, d_3)$ would be piecewise linear just like $g(d)$ was, but it is not clear how many pieces there are, and whether we can compute them easily from the structure of the row. Thus a faster algorithm for $H = 3$ (or higher) remains to be found.

5 Conclusion

In this work, we developed several algorithms that provide drastic running time improvements for the minimum cardinality problem. At this point, a couple interesting problems remain open. Does the full-matrix problem admit a FPT result if $m > 1$ but m is small (i.e., a small number of rows)? Is the full-matrix problem $W[1]$-hard in H?

References

1. Davaatseren Baatar, Natashia Boland, Sebastian Brand, and Peter J. Stuckey. Minimum cardinality matrix decomposition into consecutive-ones matrices: Cp and ip approaches. In *Proceedings of the 4th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, pages 1–15, 2007.
2. Davaatseren Baatar and Horst W. Hamacher. New lp model for multileaf collimators in radiation therapy. Contribution to the Conference ORP3, Universität Kaiserslautern, 2003.
3. Davaatseren Baatar, Horst W. Hamacher, Matthias Ehrgott, and Gerhard J. Woeginger. Decomposition of integer matrices and multileaf collimator sequencing. *Discrete Applied Mathematics*, 152(1–3):6–34, 2005.
4. Nikhil Bansal, Don Coppersmith, and Baruch Schieber. Minimizing setup and beam-on times in radiation therapy. In *Proceedings of APPROX-RANDOM*, pages 27–38, 2006.
5. T. Biedl, S. Durocher, H. Hoos, S. Luang, J. Saia, and M. Young. Improved approximations for segment minimization in intensity modulated radiation therapy, 2009. Submitted.
6. Therese Biedl, Stephane Durocher, Holger H. Hoos, Shuang Luan, Jared Saia, and Maxwell Young. A note on improving the performance of approximation algorithms for radiation therapy. *Information Processing Letters*, 111(7):326–333, 2011.
7. Sebastian Brand. The sum-of-increments constraint in the consecutive-ones matrix decomposition problem. In *Proceedings of the 24th Symposium on Applied Computing (SAC)*, pages 1417–1418, 2009.
8. Hadrien Cambazard, Eoin O'Mahony, and Barry O'Sullivan. A shortest path-based approach to the multileaf collimator sequencing problem. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR)*, pages 41–55, 2009.
9. Danny Z. Chen, Xiaobo Sharon Hu, Shuang Luan, Shahid A. Naqvi, Chao Wang, and Cedric X. Yu. Generalized geometric approaches for leaf sequencing problems in radiation therapy. In *Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC)*, pages 271–281, 2004.
10. Michael J. Collins, David Kempe, Jared Saia, and Maxwell Young. Non-negative integral subset representations of integer sets. *Information Processing Letters*, 101(3):129–133, 2007.
11. Cristian Cotrutz and Lei Xing. Segment-based dose optimization using a genetic algorithm. *Physics in Medicine and Biology*, 48(18):2987–2998, 2003.
12. Wladimir de Azevedo Pribitkin. Simple upper bounds for partition functions. *The Ramanujan Journal*, 18(1):113–119, 2009.
13. Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry (3rd edition)*. Springer-Verlag, 2008.
14. Konrad Engel. A new algorithm for optimal multileaf collimator field segmentation. *Discrete Applied Mathematics*, 152(1–3):35–51, 2005.
15. Thomas Kalinowski. The complexity of minimizing the number of shape matrices subject to minimal beam-on time in multileaf collimator field decomposition with bounded fluence. *Discrete Applied Mathematics*, 157(9):2089–2104, 2009.
16. Shuang Luan, Jared Saia, and Maxwell Young. Approximation algorithms for minimizing segments in radiation therapy. *Information Processing Letters*, 101(6):239–244, 2007.
17. Robert Sedgewick. *Algorithms in Java, Parts 1–4 (Fundamental Algorithms, Data Structures, Sorting, Searching*. Addison-Wesley, 2002.
18. R. Alfredo C. Siochi. Minimizing static intensity modulation delivery time using an intensity solid paradigm. *International Journal of Radiation Oncology * Biology * Physics*, 43(3):671–680, 1999.
19. Giulia M. G. H. Wake, Natashia Boland, and Les S. Jennings. Mixed integer programming approaches to exact minimization of total treatment time in cancer radiotherapy using multileaf collimators. *Computers and Operations Research*, 36(3):795–810, 2009.
20. Ping Xia and Lynn J. Verhey. Multileaf collimator leaf sequencing algorithm for intensity modulated beams with multiple static segments. *Medical Physics*, 25(8):1424–1434, 1998.

A Missing proof

Lemma 1. *For any segmentation $\mathcal{S}$ of a single row, there exists a compact segmentation $\mathcal{S}'$ with $|\mathcal{S}'| \leq |\mathcal{S}|$.*

Proof. Start with an arbitrary optimal segmentation $\mathcal{S}$; we can argue how to modify $\mathcal{S}$ to obtain a compact segmentation of the same size. Let i be the smallest index such that two segments S, S' of $\mathcal{S}$ begin at index i. Say S and S' have non-zero value a and a' and end at indices j and j', respectively. If $j = j'$, then the two segments could be combined into one to give a smaller segmentation, a contradiction. So $j \neq j'$, say $j < j'$.

Define two new segments S'' and S''' as follows. Segment S'' begins at i, ends at j and has value $a+a'$. Segment S''' begins at $j+1$, ends at j', and has value a'. Clearly $aS+a'S' = (a+a')S''+a'S'''$, so $\mathcal{S}' = \mathcal{S} - \{S, S'\} \cup \{S'', S'''\}$ is also an optimal segmentation, and has fewer segments that start at i. Iterate until only one segment starts at i, then iterate with all larger values where multiple segments start. (Note that all new segments in $\mathcal{S}'$ start at i or later, so this eliminates all coinciding start-indices.) Then similarly eliminate coinciding end-indices, starting at the largest one where they occur. □

B Data structures to store partitions

Recall that a partition ϕ of a value $\leq H$ is a multi-set over the universe $[H] = \{1, \ldots, H\}$. Let $t_1 > \cdots > t_\ell$ be those values that occur at least once in ϕ. We can then describe ϕ as a string

$$\sigma(\phi) = (t_1, m_{t_1}(\phi)), \ldots, t_\ell, m_{t_\ell}(\phi)),$$

where $m_{t_k}(\phi) > 0$ is the multiplicity of value t_k in ϕ, for $k = 1, \ldots, \ell$. For example, we have

$$\phi = \{4, 2, 1, 1, 1\} \Longleftrightarrow \sigma(\phi) = (4, 1, 2, 1, 1, 3)$$

A key observation is that $\sigma(\phi)$ has length $O(\sqrt{H})$. For recall that ϕ is a partition of a value $\leq H$, and hence $\sum_{k=1}^{\ell} m_{t_k}(\phi) t_k \leq H$. If we had $\ell > \sqrt{2H}$ then

$$H \geq \sum_{k=1}^{\ell} m_{t_k}(\phi) t_k \geq \sum_{k=1}^{\ell} t_k \geq \sum_{k=1}^{\ell} k = \frac{\ell(\ell+1)}{2} \geq \frac{\sqrt{2H}(\sqrt{2H}+1)}{2} > H,$$

a contradiction. So $|\sigma(\phi)| = 2\ell \leq 2\sqrt{2H} = O(\sqrt{H})$.

Thus, to store and access information about ϕ, we will store and access information about string $\sigma(\phi)$, which is a string with $O(\sqrt{H})$ entries in the alphabet $\Sigma = \{1, \ldots, H\}$. We store such strings using a *trie*, i.e., a tree where arcs to the children of a node are labelled with distinct letters from Σ. See for example [17] for more details about tries.

The node on level k of the trie refers to entry k of the strings $\sigma(\phi)$, i.e., it either distinguishes by the next value t_k for which $m_{t_k}(\phi)$ is non-zero, or (one level farther down) by what $m_{t_k}(\phi)$ is. To find the appropriate child, each node stores an array $C[1 \ldots H]$ where $C[t]$ refers to the child where the value is t.

So to find the entry for a partition ϕ (which has been stored as list $\sigma(\phi)$), we trace from the top downwards in the trie, using the kth entry in $\sigma(\phi)$ to find the appropriate child of the node on the kth level. The time to do so is $O(||\sigma(\phi)||) = O(\sqrt{H})$.

The space requirement for this trie is $O(H)$ per node. If we use a compressed trie (i.e., we only split at a node if it actually has multiple descendants), then the number of nodes in the trie is proportional to its number of leaves, which is $p(H)$. Hence the trie needs $O(p(H)\, H)$ space.

B.1 Decreasing space by increasing time

Instead of using an array to store the children of a node, we can use a binary search table or a hash-table with constant load factor. Then the space at each node is proportional to its number of children, and hence the total space used at internal nodes is $O(p(H))$. But we still need $O(p(H)\sqrt{H})$ space to store the description $\sigma(\phi)$ for all partitions ϕ, so the total space is $O(p(H)\sqrt{H})$. This savings in space comes at an increased run-time: With binary search trees, the lookup time is now $O(\log H)$ at each node, and with hash-tables, it is $O(1)$ expected time. For all but really large values of H, this rather small decrease in space does not seem to warrant the more complicated data structure and potential time-increase.

B.2 Creating partitions

We can use this trie to create all partitions of all values $\leq H$ efficiently. Let ϕ be a partition of $L \leq H$. Let t_1 be the largest value of ϕ, and let ϕ' be the partition obtained from ϕ by deleting one copy of t_1. Then ϕ' is a partition of $L - t_1$. Thus, every partition ϕ of L can be obtained by taking a partition ϕ' of a value $L' < L$ such that $L - L'$ is no smaller than the largest value in ϕ'. It is easy to see that this is a 1-1 correspondence.

to compute $\Phi(L)$, we assume that we have computed $\Phi(1), \ldots, \Phi(L-1)$ already and stored them in their appropriate tries. Create a new trie with root node r. For each $L' < L$, we obtain the partitions of L with largest value $L - L'$ by scanning the trie that stores $\Phi(L')$. More precisely, ignore all partitions in $\Phi(L')$ that are located at children $C[L - L' + 1, \ldots, L']$ of the root; these have largest value bigger than $L - L'$. Then scan through each remaining partition of L', add one value $L - L'$ to it to obtain a partition of L, and add it into the trie that stores $\Phi(L)$. This takes $O(\sqrt{H})$ time per partition that is inserted, and hence $O(p(L)\sqrt{H})$ time overall. Doing this for $L = 1, \ldots, H$ finds all partitions of H in time $O(\sqrt{H}(p(1) + \cdots + p(H))) = O(p(H)\sqrt{H})$.