A log-linear time algorithm for constrained changepoint detection

Toby Dylan Hocking (toby.hocking@r-project.org)
Guillem Rigaill (guillem.rigaill@inra.fr)
Paul Fearnhead (p.fearnhead@lancaster.ac.uk)
Guillaume Bourque (guil.bourque@mcgill.ca)

March 10, 2017

Abstract

Changepoint detection is a central problem in time series and genomic data. For some applications, it is natural to impose constraints on the directions of changes. One example is ChIP-seq data, for which adding an up-down constraint improves peak detection accuracy, but makes the optimization problem more complicated. We show how a recently proposed functional pruning technique can be adapted to solve such constrained changepoint detection problems. This leads to a new algorithm which can solve problems with arbitrary affine constraints on adjacent segment means, and which has empirical time complexity that is log-linear in the amount of data. This algorithm achieves state-of-the-art accuracy in a benchmark of several genomic data sets, and is orders of magnitude faster than existing algorithms that have similar accuracy. Our implementation is available as the PeakSegPDPA function in the coseg R package, `https://github.com/tdhock/coseg`

Contents

1 Introduction

Changepoint detection is a central problem in fields such as finance or genomics, where n data are gathered in a sequence over time or space. Many models define the optimal changepoints using maximum likelihood, resulting in a discrete optimization problem. Multiple changepoint detection models seek the optimal K segments ($K-1$ changes), which amounts to optimizing likelihood parameters over a space that contains $O(n^{K-1})$ discrete arrangements of changepoints. In general this problem can be solved in $O(Kn^2)$ time using the original dynamic programming algorithm of Auger and Lawrence [1989]. Recently proposed pruning techniques reduce the number of changepoints considered by the algorithm, thus reducing time complexity to $O(Kn \log n)$ while maintaining optimality [Rigaill, 2010, Johnson, 2013, Maidstone et al., 2016].

In "unconstrained" changepoint models, there are no contraints between model parameters on separate segments. To regularize and obtain a more interpretable model, it is often desirable to introduce constraints between model parameters before and after changepoints. For example, the main problem that motivates this paper is peak detection in ChIP-seq data, which provide noisy measurements of protein binding or modification throughout a genome [Bailey et al., 2013]. An up-down constrained changepoint detection model has been shown to achieve state-of-the-art peak detection accuracy in ChIP-seq data [Hocking et al., 2015]. The constraints of this model force an up change in the segment mean parameter after each down change, and vice versa. The fastest existing solver for this problem is the Constrained Dynamic Programming Algorithm (CDPA), which has two issues. First, it is a heuristic algorithm that is not guaranteed to recover the optimal solution. Second, its $O(Kn^2)$ quadratic time complexity is too slow for use on large data sets. In this paper we propose a new algorithm that fixes both of these issues.

1.1 Contributions and organization

We begin by discussing previous research into pruning techniques for solving unconstrained changepoint detection problems (Section 2), then state the constrained optimization problems (Section 3). Our main contribution is Section 4, which generalizes the functional pruning technique of Rigaill [2010], thus providing a new Generalized Pruned Dynamic Progamming Algorithm (GPDPA) for solving a class of constrained changepoint detection problems. We show that the GPDPA achieves state-of-the-art speed and accuracy in genomic data with several different labeled patterns (Section 5), then conclude by discussing the significance of our contributions (Section 6).

2 Related work

There are many efficient algorithms available for computing the optimal $K-1$ changepoints in n data points. Auger and Lawrence [1989] proposed an $O(Kn^2)$ algorithm for computing the sequence of models with $1, \ldots, K$ segments. Jackson et al. [2005] consider a related approach, which introduces a penalty for each changepoint, rather than fixing the number of changepoints. Their $O(n^2)$ algorithm computes the single model for a given penalty constant λ. Both of these algorithms recover the optimal solution, and follow from using dynamic programming updates [Bellman, 1961] to recursively compute the maximum likelihood from 1 to n data points. Alternatively there are methods which are

	No pruning	Functional pruning
Unconstrained	Dynamic Programming Algorithm (DPA) Optimal solution, $O(Kn^2)$ time Auger and Lawrence [1989]	Pruned DPA (PDPA) Optimal solution, $O(Kn \log n)$ time Rigaill [2010], Johnson [2013]
Up-down constrained	Constrained DPA (CDPA) Sub-optimal solution, $O(Kn^2)$ time Hocking et al. [2015]	Generalized Pruned DPA (GPDPA) Optimal solution, $O(Kn \log n)$ time **This paper**

Table 1: Our contribution is the Generalized Pruned Dynamic Programming Algorithm (GPDPA), which uses a functional pruning technique to compute the constrained optimal $K-1$ changepoints in a sequence of n data, in $O(Kn \log n)$ time on average.

computationally faster but are not guaranteed to find the optimal segmentation. The most popular of these is the binary segmentation algorithm which has $O(Kn)$ worst-case time complexity [Scott and Knott, 1974]. An L1 relaxation of this problem is known as the fused lasso signal approximator, for which efficient solvers also exist [Hoefling, 2010].

Several pruning methods have been recently proposed in order to reduce time complexity, while maintaining optimality. Rigaill [2010] and Johnson [2011] independently discovered a functional pruning technique, which results in algorithms with $O(n \log n)$ average time complexity. Killick et al. [2011] proposed an inequality pruning technique, which results in an algorithm with average time complexity from $O(n)$ to $O(n^2)$, depending on the number of changes. Maidstone et al. [2016] provides a clear discussion on the differences between the two pruning techniques.

All algorithms discussed thus far are for solving problems with no constraints between adjacent segment mean parameters, but there are many examples of constrained changepoint detection models. Rather than searching all possible changepoints and likelihood parameters, the idea is to use a constraint in order to search a smaller, more interpretable model space. For example, Haiminen et al. [2008] propose an $O(Kn^2)$ algorithm for unimodal regression, which enforces no up changes after the first down change. Hocking et al. [2015] proposed an $O(Kn^2)$ algorithm for peak detection, which enforces a down change after each up change, and vice versa.

Isotonic regression is another example of a constrained changepoint detection model. There is no limit on the number of segments K, but the segment means are constrained to be non-decreasing. This problem can be solved in $O(n)$ time using the pool-adjacent-violators algorithm [Mair et al., 2009], or in $O(n \log n)$ time using a dynamic programming algorithm [Rote, unpublished]. An L1 relaxation of this problem is known as nearly-isotonic regression [Tibshirani et al., 2011]. A problem known as reduced isotonic regression occurs by imposing an additional constraint of K segments [Schell and Singh, 1997]. The techniques for solving this problem lead to sub-quadratic time algorithms [Hardwick and Stout, 2014], but do not generalize to other kinds of constraints (such as unimodal regression or peak detection).

Our contribution in this paper is proving that the functional pruning technique can be generalized to constrained changepoint models (Table 1). Our resulting Generalized Pruned Dynamic Programming Algorithm (GPDPA) enjoys $O(Kn \log n)$ time complexity, and works for any changepoint model with affine constraints between adjacent segment means (including isotonic regression, unimodal regression, and peak detection).

3 Isotonic regression and changepoint models

Although our proposed algorithm can solve many constrained changepoint detection problems (Section 4.5), we will simplify our discussion by emphasizing the isotonic regression model.

3.1 Classical isotonic regression

The classical isotonic regression model is defined as the most likely sequence of non-decreasing segment means. More precisely, assume that the data $\mathbf{y} \in \mathbb{R}^n$ are a realization of a probability distribution with mean parameter $\mathbf{m} \in \mathbb{R}^n$. For example, assuming $y_t \sim \mathcal{N}(m_t, \sigma^2)$ and performing maximum likelihood inference results in a convex minimization problem with affine constraints,

$$\begin{aligned} &\underset{\mathbf{m} \in \mathbb{R}^n}{\text{minimize}} \quad \sum_{t=1}^{n} \ell(y_t, m_t) \\ &\text{subject to} \quad m_t \leq m_{t+1}, \, \forall t < n. \end{aligned} \tag{1}$$

The convex loss function $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ in the case of the Gaussian likelihood is the square loss $\ell(y, m) = (y - m)^2$. This optimization problem (1) is referred to as isotonic regression, and can be efficiently solved in $O(n)$ time using the Pool-Adjacent-Violators Algorithm (PAVA) [Best and Chakravarti, 1990].

Since isotonic regression imposes no limit on the number of changepoints ($m_t < m_{t+1}$), it tends to overfit. For example, consider the toy data set $\mathbf{y} = \begin{bmatrix} 2 & 5 & 30 & 34 & 600 & 621 \end{bmatrix} \in \mathbb{R}^6$. Because these data are strictly increasing, the isotonic regression (1) solution is the trivial model $m_t = y_t$. However, these data contain only two large changes. To recover these changes, we could instead use the segment neighborhood model, which we discuss in the next section.

3.2 Segment neighborhood changepoint model

The segment neighborhood model of Auger and Lawrence [1989] uses the same cost function as isotonic regression, but a different constraint set. There is no constraint on the direction of changes, but there must be exactly $K \leq n$ distinct segments ($K-1$ changes).

$$\begin{aligned} &\underset{\mathbf{m}\in\mathbb{R}^n}{\text{minimize}} \ \sum_{t=1}^{n} \ell(y_t, m_t) \\ &\text{subject to} \ \sum_{t=1}^{n-1} I(m_t \neq m_{t+1}) = K-1. \end{aligned} \tag{2}$$

This optimization problem is non-convex since the model complexity is the number of changepoints, measured via the non-convex indicator function I. Nonetheless, the optimal solution can be computed in $O(Kn^2)$ time using the standard dynamic programming algorithm [Auger and Lawrence, 1989]. By exploiting the structure of the convex loss function ℓ, the pruned dynamic programming algorithm of Rigaill [2010] computes the same optimal solution in faster $O(Kn\log n)$ time.

Unlike isotonic regression, the segment neighborhood model does not constrain the direction of the changes. Thus, for some data sets $\mathbf{y}$, the segment neighborhood model may recover a change down ($m_t > m_{t+1}$). For applications where isotonic regression is used, it would be desirable to compute a model with K non-decreasing segment means. This results in the reduced isotonic regression problem, which we introduce in the next section.

3.3 Reduced isotonic regression

The idea of fitting a non-decreasing function with a limited number of changepoints has been previously described as reduced isotonic regression [Schell and Singh, 1997]. Combining the constraints of the isotonic regression (1) and segment neighborhood (2) problems gives

$$\begin{aligned} &\underset{\mathbf{m}\in\mathbb{R}^n}{\text{minimize}} \ \sum_{t=1}^{n} \ell(y_t, m_t) \\ &\text{subject to} \ \sum_{t=1}^{n-1} I(m_t \neq m_{t+1}) = K-1, \\ &\qquad m_t \leq m_{t+1}, \ \forall t < n. \end{aligned} \tag{3}$$

In the next section, we explain how functional pruning can be used for solving this and related changepoint problems.

4 Functional pruning algorithms for constrained changepoint models

We begin by discussing an algorithm for solving the reduced isotonic regression problem, then explain how the algorithm generalizes to other constrained changepoint problems.

4.1 Equivalent optimization space

The reduced isotonic regression problem (3) has n segment mean variables m_t, one for each data point t. To derive our algorithm, we re-write the problem in terms of the mean $u_k \in \mathbb{R}$ and endpoint $t_k \in \{1,\dots,n\}$ for each segment $k \in \{1,\dots,K\}$.

Definition 1 (Reduced isotonic regression optimization space)**.** *Let $(\mathbf{u},\mathbf{t}) \in \mathcal{I}_K^n$ be the set of non-decreasing segment means $u_1 \leq \cdots \leq u_K$ and increasing changepoint indices $0 = t_0 < t_1 < \cdots < t_{K-1} < t_K = n$.*

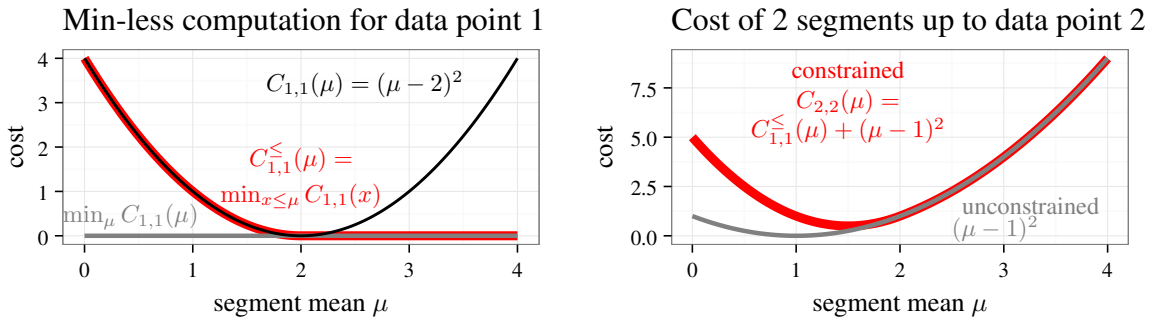

Figure 1: Comparison of previous unconstrained algorithm (grey) with new algorithm that constrains segment means to be non-decreasing (red), for the toy data set $\mathbf{y} = [2, 1, 0, 4] \in \mathbb{R}^4$ and the square loss. **Left:** rather than computing the unconstrained minimum (constant grey function), the new algorithm computes the min-less operator (red), resulting in a larger cost when the segment mean is less than the first data point ($\mu < 2$). **Right:** adding the cost of the second data point $(\mu - 1)^2$ and minimizing yields equal means $u_1 = u_2 = 1.5$ for the constrained model and decreasing means $u_1 = 2$, $u_2 = 1$ for the unconstrained model.

Each segment mean u_k is assigned to data points $\tau \in (t_{k-1}, t_k] \subset \{1, \ldots, n\}$, resulting in the following cost for each segment $k \in \{1, \ldots, K\}$,

$$h_{t_{k-1},t_k}(u_k) = \sum_{\tau=t_{k-1}+1}^{t_k} \ell(y_\tau, u_k). \tag{4}$$

The reduced isotonic regression problem can be equivalently written as

$$\underset{(\mathbf{u},\mathbf{t}) \in \mathcal{I}^n_K}{\text{minimize}} \sum_{k=1}^{K} h_{t_{k-1},t_k}(u_k) \tag{5}$$

Rather than explicitly summing over data points i as in problem (3), this problem uses the equivalent sum over segments k.

4.2 Dynamic programming update rules

Optimization problem (5) has K segment mean variables u_k and $K - 1$ changepoint index variables t_k. Minimizing over all variables except the last segment mean u_K results in the following definition of the optimal cost.

Definition 2 (Optimal cost with last segment mean μ)**.** *Let $C_{K,n}(\mu)$ be the optimal cost of the segmentation with K segments, up to data point n, with last segment mean μ:*

$$C_{K,n}(\mu) = \min_{(\mathbf{u},\mathbf{t}) \in \mathcal{I}^n_K \mid u_K = \mu} \left\{ \sum_{k=1}^{K} h_{t_{k-1},t_k}(u_k) \right\}. \tag{6}$$

As in the PDPA of Rigaill [2010], our proposed dynamic programming algorithm uses an exact representation of the $C_{k,t} : \mathbb{R} \to \mathbb{R}$ cost functions. Each $C_{k,t}(\mu)$ is represented as a piecewise function on intervals of μ. This is implemented as a linked list of FunctionPiece objects in C++ (for details see Section B). Each element of the linked list represents a convex function piece, and implementation details depend on the choice of the loss function ℓ (for an example using the square loss see Section 4.3).

In the original unconstrained PDPA, computing the $C_{k,t}(u_k)$ function requires taking the minimum of $C_{k,t-1}(u_k)$ (a function of the last segment mean u_k) and $\hat{C}_{k-1,t-1} = \min_{u_{k-1}} C_{k-1,t-1}(u_{k-1})$ (the constant loss resulting from an unconstrained minimization with respect to the previous segment mean u_{k-1}). The main novelty of our paper is the discovery that this update can also be computed efficiently for constrained problems. For example in reduced isotonic regression the second term is no longer a constant, but instead a function of u_k, $C^{\leq}_{k-1,t-1}(u_k) = \min_{u_{k-1} \leq u_k} C_{k-1,t-1}(u_{k-1})$, which we refer to as the min-less operator (Figure 1, left).

Definition 3 (Min-less operator)**.** *Given any real-valued function $f : \mathbb{R} \to \mathbb{R}$, we define the min-less operator of that function as $f^{\leq}(\mu) = \min_{x \leq \mu} f(x)$.*

The min-less operator is used in the following Theorem, which states the update rules used in our proposed algorithm.

Theorem 1 (Generalized Pruned Dynamic Programming Algorithm for reduced isotonic regression)**.** *The optimal cost functions $C_{k,t}$ can be recursively computed using the following update rules.*

1. *For $k = 1$ we have $C_{1,1}(\mu) = \ell(y_1, \mu)$, and for the other data points $t > 1$ we have*

$$C_{1,t}(\mu) = C_{1,t-1}(\mu) + \ell(y_t, \mu) \tag{7}$$

2. *For $k > 1$ and $t = k$ we have*

$$C_{k,k}(\mu) = \ell(y_k, \mu) + C^{\leq}_{k-1,k-1}(\mu) \tag{8}$$

3. *In all other cases we have*

$$C_{k,t}(\mu) = \ell(y_t, \mu) + \min\{C^{\leq}_{k-1,t-1}(\mu),\, C_{k,t-1}(\mu)\}. \tag{9}$$

Proof. Case 1 and 2 follow from Definition 2, and there is a proof for case 3 in Section A. □

The dynamic programming algorithm requires computing $O(Kn)$ cost functions $C_{k,t}$. As in the original pruned dynamic programming algorithm, the time complexity of the algorithm is $O(KnI)$ where I is the number of intervals (convex function pieces; candidate changepoints) that are used to represent the cost functions. The theoretical maximum number of intervals is $I = O(n)$, implying a time complexity of $O(Kn^2)$ [Rigaill, 2015]. However, this maximum is only achieved in pathological synthetic data sets, such as a monotonic increasing data sequence. The average number of intervals in real data sets is empirically $I = O(\log n)$, as we will show in Section 5.1. Thus the average time complexity of the algorithm is $O(Kn \log n)$.

4.3 Example and comparison with unconstrained case

To clarify the discussion, consider the toy data set $\mathbf{y} = \begin{bmatrix} 2 & 1 & 0 & 4 \end{bmatrix} \in \mathbb{R}^4$ and the square loss $\ell(y, \mu) = (y - \mu)^2$. The first step of the algorithm is to compute the minimum and the maximum of the data (0,4) in order to bound the possible values of the segment mean μ. Then the algorithm computes the optimal cost in $k = 1$ segment up to data point $t = 1$:

$$C_{1,1}(\mu) = (2 - \mu)^2 = 4 - 4\mu + \mu^2 \text{ (for } \mu \in [0, 4]) \tag{10}$$

This function can be stored for all values of μ via the three real-valued coefficients (constant $= 4$, linear $= -4$, quadratic $= 1$). To compute the optimal cost in $K = 2$ segments, we first compute the min-less operator (red curve on left of Figure 1),

$$C^{\leq}_{1,1}(\mu) = \begin{cases} 4 - 4\mu + \mu^2 & \text{if } \mu \in [0, 2],\ \mu' = \mu, \\ 0 + 0\mu + 0\mu^2 & \text{if } \mu \in [2, 4],\ \mu' = 2. \end{cases} \tag{11}$$

This function can be stored as a list of two intervals of μ values, each with associated real-valued coefficients. In addition, to facilitate recovery of the optimal parameters, we store the previous segment mean μ' and endpoint (not shown). Note that $\mu' = \mu$ means that the equality constraint is active ($u_1 = u_2$).

By adding the first min-less function $C^{\leq}_{1,1}(\mu)$ to the cost of the second data point $(\mu - 2)^2$ we obtain the optimal cost in $K = 2$ segments up to data point $t = 2$,

$$C_{2,2}(\mu) = \begin{cases} 5 - 6\mu + 2\mu^2 & \text{if } \mu \in [0, 2],\ \mu' = \mu, \\ 1 - 2\mu + 1\mu^2 & \text{if } \mu \in [2, 4],\ \mu' = 2. \end{cases} \tag{12}$$

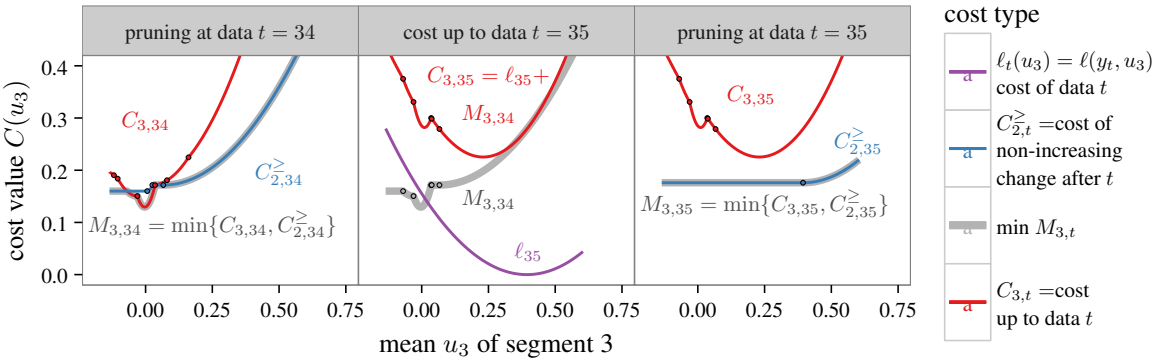

Figure 2: Demonstration of GPDPA for the PeakSeg model (13) with $k = 3$ segments. Cost functions are stored as piecewise functions on intervals (black dots show limits between function pieces). **Left:** the min $M_{3,34}$ is the minimum of two functions: $C^{\geq}_{2,34}$ is the cost if the second segment ends at data point $t = 34$ (the min-more operator forces a non-increasing change after), and $C_{3,34}$ is the cost if the second segment ends before that. **Middle:** the cost $C_{3,35}$ is the sum of the min $M_{3,34}$ and the cost of the next data point ℓ_{35}. **Right:** in the next step, all previously considered changepoints are pruned (cost $C_{3,35}$), since the model with a the second segment ending at data point $t = 35$ is always less costly ($C^{\geq}_{2,35}$).

Note that the minimum of this function is achieved at $\mu = 1.5$ which occurs in the first of the two function pieces (red curve on right of Figure 1), with an equality constraint active. This implies the optimal model up to data point $t = 2$ with $k = 2$ non-decreasing segment means actually has no change ($u_1 = u_2 = 1.5$). In contrast, the minimum of the cost computed by the unconstrained algorithm is at $u_2 = 1$ (grey curve on right of Figure 1), resulting in a change down from $u_1 = 2$.

4.4 The PeakSeg up-down constraint

The PeakSeg model described by Hocking et al. [2015] is the most likely segmentation where the first change is up, all up changes are followed by down changes, and all down changes are followed by up changes. More precisely, the constrained optimization problem can be stated as

$$\begin{aligned} \underset{\substack{\mathbf{u}\in\mathbb{R}^K \\ 0=t_0<t_1<\cdots<t_{K-1}<t_K=n}}{\text{minimize}} \quad & \sum_{k=1}^{K} h_{t_{k-1},t_k}(u_k) \\ \text{subject to} \quad & u_{k-1} \leq u_k \; \forall k \in \{2, 4, \dots\}, \\ & u_{k-1} \geq u_k \; \forall k \in \{3, 5, \dots\}. \end{aligned} \tag{13}$$

Our proposed Generalized Pruned Dynamic Programming Algorithm (GPDPA) can be used to solve the PeakSeg problem. The initialization $k = 1$ is the same as in the reduced isotonic regression solver (Section 4.2). The dynamic programming updates for even $k \in \{2, 4, \dots\}$ are also the same. However, to constrain non-increasing changes, the updates for odd $k \in \{3, 5, \dots\}$ are

$$C_{k,t}(\mu) = \ell(y_t, \mu) + \min\{C^{\geq}_{k-1,t-1}(\mu), C_{k,t-1}(\mu)\}, \tag{14}$$

where the min-more operator is defined for any function $f : \mathbb{R} \to \mathbb{R}$ as $f^{\geq}(\mu) = \min_{x \geq \mu} f(x)$. Figure 2 shows the geometric interpretation of the min-more operator, along with an example of how the $\min\{\}$ operation performs pruning. We implemented this algorithm using the Poisson loss $\ell(y, \mu) = \mu - y \log \mu$, since our application in Section 5 is on count data $y \in \mathbb{Z}_+ = \{0, 1, 2, \dots\}$. We implemented this algorithm in C++, and our free/open-source code is available as the PeakSegPDPA function in the coseg R package for constrained optimal segmentation (`https://github.com/tdhock/coseg`). Implementation details can be found in Section B.

4.5 General affine inequality constraints between adjacent segment means

In this section we briefly discuss how our proposed Generalized Pruned Dynamic Programming Algorithm (GPDPA) can be used to solve any optimization problem with affine inequality constraints between adjacent segment means. For each change $k \in \{1, \dots, K-1\}$, let $a_k, b_k, c_k \in \mathbb{R}$ be arbitrary coefficients that define affine functions $g_k(u_k, u_{k+1}) = a_k u_k + b_k u_{k+1} + c_k$. The changepoint detection problem with general affine constraints is

$$\begin{aligned} \underset{\substack{\mathbf{u}\in\mathbb{R}^K \\ 0=t_0<t_1<\cdots<t_{K-1}<t_K=n}}{\text{minimize}} \quad & \sum_{k=1}^{K} h_{t_{k-1},t_k}(u_k) \\ \text{subject to} \quad & \forall k \in \{1, \dots, K-1\}, \\ & g_k(u_k, u_{k+1}) \leq 0. \end{aligned} \tag{15}$$

Some examples of models that are special cases:

1. If we take all $a_k, b_k, c_k = 0$ then the constraints are trivially satisfied, we recover the unconstrained segment neighborhood problem (2).
2. If we take all $a_k = 1$, $b_k = -1$ and $c_k = 0$ we recover the reduced isotonic regression problem (5).
3. For the PeakSeg problem (13), we take all $c_k = 0$. For odd $k \in \{1, 3, \dots\}$ we take $a_k = 1$, $b_k = -1$ and for even $k \in \{2, 4, \dots\}$ we take $a_k = -1$, $b_k = 1$.

To solve these problems, we need to compute the analog of the min-less/more operator, which we call the constrained minimization operator. For any cost function $f : \mathbb{R} \to \mathbb{R}$ and constraint function $g : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, we define the constrained minimization operator $f^g : \mathbb{R} \to \mathbb{R}$ as

$$f^g(u_k) = \min_{u_{k-1}:g(u_{k-1},u_k)\leq 0} f(u_{k-1}). \tag{16}$$

When g is affine, the constrained minimization operator is either non-decreasing or non-increasing. In this case it can be computed using a simple algorithm that scans the piecewise function f either from left to right or right to left. When a local minimum is found, its value is recorded, and a constant function piece is added (for details see pseudocode for MinLess algorithm in Section B.2). The constrained minimization operator is used in the following general dynamic programming update rule which can be used to compute the solution to (15)

$$C_{k,t}(\mu) = \ell(y_t, \mu) + \min\{C_{k,t-1}(\mu),\, C^{g_{k-1}}_{k-1,t-1}(\mu)\}. \tag{17}$$

We note that this update rule is valid for constraint functions g more general than affine functions. However, the closed-form computation of the constrained minimization operator (16) would possibly be much more difficult for these more general constraint functions (e.g. quadratic constraint functions).

5 Results on peak detection in ChIP-seq data

The real data analysis problem that motivates this work is the detection of peaks in ChIP-seq data [Bailey et al., 2013], which are typically represented as a vector of non-negative counts $\mathbf{y} \in \mathbb{Z}^n_+$ of aligned sequence reads for n continguous bases in a genome. Data sizes are between $n = 10^5$ (maximum of the benchmark we consider) and $n = 10^8$ (largest region with no gaps in the human genome hg19). A peak detector can be represented as a function $c(\mathbf{y}) \in \{0, 1\}^n$ for binary classification at every base position. The positive class is peaks (genomic regions with large values, representing protein binding or modification) and the negative class is background noise (small values).

In the supervised learning framework of Hocking et al. [2016], a data set consists of m count data vectors $\mathbf{y}_1, \dots, \mathbf{y}_m$ along with labels $L_1, \dots, L_m$ that identify regions with and without peaks. Briefly, the number of errors $E[c(\mathbf{y}_i), L_i]$ is the total of false positives (negative labels with a predicted peak) plus false negatives (positive labels with no predicted peak). The benchmark consists of seven histone ChIP-seq data sets, each with a different peak

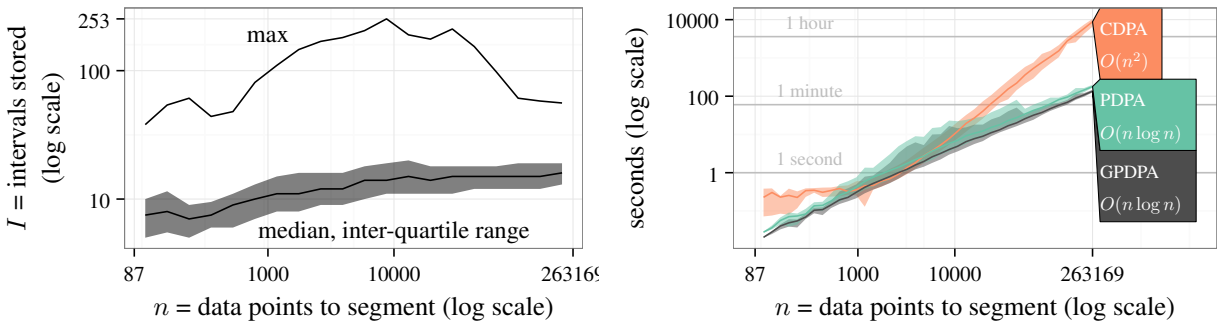

Figure 3: Empirical speed analysis on 2752 count data vectors from the histone mark ChIP-seq benchmark. For each vector we ran the GPDPA with the up-down constraint and a max of $K = 19$ segments. The expected time complexity is $O(KnI)$ where I is the average number of intervals (function pieces; candidate changepoints) stored in the $C_{k,t}$ cost functions. **Left**: number of intervals stored is $I = O(\log n)$ (median, inter-quartile range, and maximum over all data points t and segments k). **Right**: time complexity of the GPDPA is $O(n \log n)$ (median line and min/max band).

pattern (experiment type, labeler, cell types). The goal in each data set is to learn the pattern encoded in the labels, and find a classifier c that minimizes the total number of incorrectly predicted labels in a held-out test set:

$$\underset{c}{\text{minimize}} \sum_{i=1}^{m} E\left[c(\mathbf{y}_i), L_i\right]. \tag{18}$$

Hocking et al. [2015] proposed a constrained dynamic programming algorithm (CDPA) to approximately compute the optimal changepoints, subject to the PeakSeg up-down constraint (Section 4.4). The CDPA has been shown to achieve state-of-the-art peak detection accuracy, by classifying even-numbered segments k as peaks, and odd-numbered segments k as background noise. However, its quadratic $O(Kn^2)$ time complexity makes it too slow to run on large ChIP-seq data sets.

In this section, we show that our proposed GPDPA can be used to overcome this speed drawback, while maintaining state-of-the-art accuracy. To show the importance of enforcing the up-down constraint, we consider the unconstrained Pruned Dynamic Programming Algorithm (PDPA) of Rigaill [2010] as a baseline (Table 1). We also compare against two popular heuristics from the bioinformatics literature, in order to demonstrate that constrained optimization algorithms such as the CDPA and GPDPA are more accurate.

5.1 Empirical time complexity in ChIP-seq data

The ChIP-seq benchmark consists of seven labeled histone data sets. Overall there are 2752 count data vectors $\mathbf{y}_i$ to segment, varying in size from $n = 87$ to $n = 263169$ data. For each count data vector $\mathbf{y}_i$, we ran each algorithm (CDPA, PDPA, GDPDA) with a maximum of $K = 19$ segments. This implies a maximum of 9 peaks (one for each even-numbered segment), which is more than enough in these relatively small data sets. To analyze the empirical time complexity, we recorded the number of intervals stored in the $C_{k,t}$ cost functions (Section 4), as well as the computation time in seconds.

As in the PDPA, the time complexity of our proposed GPDPA is $O(KnI)$, which depends on the number of intervals I (candidate changepoints) stored in the $C_{k,t}$ cost functions [Rigaill, 2015]. We observed that the number of intervals stored by the GPDPA increases as a sub-linear function of the number of data points n (left of Figure 3). For the largest data set ($n = 263169$), the algorithm only stored median=16 and maximum=43 intervals. The most intervals stored was 253 for one data set with $n = 7776$. These results suggest that our proposed GPDPA only stores on average $O(\log n)$ intervals (possible changepoints), as in the original PDPA. The overall empirical time complexity is thus $O(Kn \log n)$ for K segments and n data points.

We recorded the timings of each algorithm for computing models with up to $K = 19$ segments (a total of 10 peak models $k \in \{1, 3, \ldots, 19\}$, from 0 to 9 peaks). Since K is constant, the expected time complexity was $O(n^2)$ for the

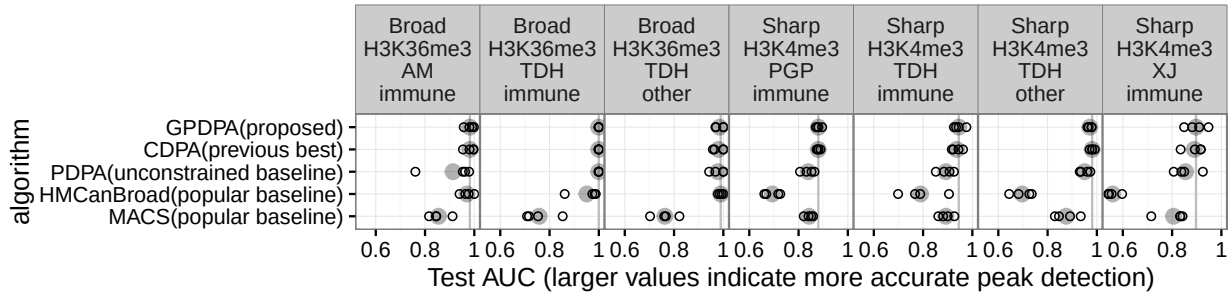

Figure 4: Four-fold cross-validation was used to estimate peak detection accuracy. Each panel shows one of seven ChIP-seq data sets, labeled by experiment (Broad H3K36me3), labeler (AM), and cell types (immune). Each black circle shows test AUC in one of four cross-validation folds, the shaded grey circle is the mean, and the vertical line is the maximum mean in each data set. It is clear that the proposed GPDPA is just as accurate as the previous state-of-the-art CDPA, and both are more accurate than the other baseline methods.

CDPA and $O(n \log n)$ for the PDPA and GPDPA. In agreement with these expectations, our proposed GPDPA shows $O(n \log n)$ asymptotic timings similar to the PDPA (right of Figure 3).

It is clear that the $O(n^2)$ CDPA algorithm is slower than the other two algorithms, especially for larger data sets. For the largest count data vector ($n = 263169$), the CDPA took over two hours, but the GPDPA took only about two minutes. Our proposed GPDPA is nearly as fast as MACS [Zhang et al., 2008], a heuristic from the bioinformatics literature which took about 1 minute to compute 10 peak models for this data set.

The total computation time to process all 2752 count data vectors was 156 hours for the CDPA, and only 6 hours for the GPDPA (26 times faster). Overall, these results suggest that our proposed GPDPA enjoys $O(n \log n)$ time complexity in ChIP-seq data, which makes it possible to use for very large data sets.

5.2 Test accuracy in ChIP-seq data

For the optimal changepoint detection algorithms (CDPA, PDPA, GPDPA), the prediction problem simplifies to selecting the number of segments $K_i \in \{1, 3, \ldots, 19\}$ for each data vector i, resulting in a predicted peak vector $c^{K_i}(\mathbf{y}_i) \in \{0, 1\}^n$. We select the number of segments using an oracle penalty $K_i^\lambda = \arg\min_k l_{ik} + \lambda o_{ik}$ [Cleynen and Lebarbier, 2014], where l_{ik} is the Poisson loss and o_{ik} is the oracle model complexity for the model with k segments for data vector i. The problem thus simplifies to learning a scalar penalty constant λ,

$$\underset{\lambda}{\text{minimize}} \sum_{i=1}^{m} E\left[c^{K_i^\lambda}(\mathbf{y}_i), L_i\right]. \tag{19}$$

To demonstrate that changepoint detection algorithms are more accurate than typical heuristics from the bioinformatics literature, we also compared with the MACS and HMCanBroad methods [Zhang et al., 2008, Ashoor et al., 2013]. MACS is a popular heuristic for data with a sharp peak pattern such as H3K4me3, and HMCanBroad is a popular heuristic for data with a broad peak pattern such as H3K36me3. Although they are not designed for supervised learning, we trained them by performing grid search over a single significance threshold parameter (qvalue for MACS and finalThreshold for HMCanBroad).

In each of the seven data sets in the histone benchmark, we performed four-fold cross-validation and computed test AUC (area under the Receiver Operating Characteristic curve) to estimate the accuracy of each algorithm. The previous algorithm with state-of-the-art accuracy on this benchmark was the CDPA, which enforces the up-down constraint on segment means. We expected our proposed GPDPA to perform just as well, since it also enforces that constraint. In agreement with our expectation, we observed that the CDPA and GPDPA yield comparable test AUC in all seven data sets (Figure 4). In contrast, the unconstrained PDPA had much lower test AUC in several data sets, because of lower true positive rates. These results provide convincing evidence that the constraint is necessary for optimal peak detection accuracy.

Since the baseline HMCanBroad algorithm was designed for data with a broad peak pattern, we expected it to perform well in the H3K36me3 data. In agreement with this expectation, HMCanBroad showed state-of-the-art test AUC in two H3K36me3 data sets (broad peak pattern), but was very inaccurate in four H3K4me3 data sets (sharp peak pattern). We expected the baseline MACS algorithm to perform well in the H3K4me3 data sets, since it was designed for data with a sharp peak pattern. In contrast to this expectation, MACS had test AUC values much lower than the optimization-based algorithms in all seven data sets (Figure 4). These results suggest that constrained optimal changepoint detection algorithms are more accurate than the heuristics from the bioinformatics literature.

6 Discussion and conclusions

Algorithms for changepoint detection can be classified in terms of time complexity, optimality, constraints, and pruning techniques (Table 1). In this paper, we investigated generalizing the functional pruning technique originally discovered by Rigaill [2010] and Johnson [2011]. We showed that the functional pruning technique can be used to compute optimal changepoints subject to affine constraints on adjacent segment mean parameters.

We showed that our proposed Generalized Pruned Dynamic Programming Algorithm (GPDPA) enjoys the same log-linear $O(Kn \log n)$ time complexity as the original unconstrained PDPA, when applied to peak detection in ChIP-seq data sets (Figure 3). However, we observed that the up-down constrained GPDPA is much more accurate than the unconstrained PDPA (Figure 4). These results suggest that the up-down constraint is necessary for computing a changepoint model with optimal peak detection accuracy. Indeed, we observed that the GPDPA enjoys the same state-of-the-art accuracy as the previous best, the relatively slow quadratic $O(Kn^2)$ time CDPA.

We observed that the heuristic algorithms which are popular in the bioinformatics literature (MACS, HMCanBroad) are much less accurate than the optimal changepoint detection algorithms (CDPA, PDPA, GPDPA). In the past these sub-optimal heuristics have been preferred because of their speed. For example, the CDPA took 2 hours to compute 10 peak models in the largest data set in the ChIP-seq benchmark, whereas the GPDPA took 2 minutes, and the MACS heuristic took 1 minute. Using our proposed GPDPA, it is now possible to compute highly accurate models in an amount of time that is comparable to heuristic algorithms. Our proposed GPDPA can now be used as an optimal alternative to heuristic algorithms, even for large data sets.

For future work we will be interested in exploring pruning techniques for other constrained changepoint models. When the number of expected changepoints grows with the number of data points, then $K = O(n)$ and our proposed GPDPA has $O(n^2 \log n)$ average time complexity (since it computes all models with $1, \ldots, K$ segments). We have already started modifying the GPDPA for optimal partitioning [Jackson et al., 2005], which results in the Generalized Functional Prunining Optimal Partitioning (GFPOP) algorithm (Section B.5). It computes the K-segment model for a single penalty constant λ (without computing models with $1, \ldots, K-1$ segments) in $O(n \log n)$ time.

7 Reproducible Research Statement

The source code and data used to create this manuscript (including all figures) is available at `https://github.com/tdhock/PeakSegFPOP-paper`

8 Acknowledgements

This work was supported by a Discovery Frontiers project grant, "The Cancer Genome Collaboratory," jointly sponsored by the Natural Sciences and Engineering Research Council (NSERC), Genome Canada (GC), the Canadian Institutes of Health Research (CIHR) and the Canada Foundation for Innovation (CFI).

The supplementary materials begin on this page.

A Proof of optimality of dynamic programming algorithm

In this section we give a proof of Theorem 1.

Proof. Case 1 and 2 follow from the definition of $C_{K,t}(u)$.

We now focus on case 3. First notice that by definition of $C_{K,t+1}(u)$ (i.e. the optimal segmentation) we must have $C_{K,t+1}(u) \leq C_{K,t}(u) + \ell(y_t, u)$ and also $C_{K,t+1}(u) \leq C_{K-1,t}(u) + \ell(y_t, u)$. Thus we have $C_{K,t+1}(u) \leq \min\{C_{K,t}(u), C_{K-1,t}(u)\} + \ell(y_{t+1}, u)$.

Now let us assume,

$$C_{K,t+1}(u) < \min\{C_{K,t}(u), C_{K-1,t}(u)\} + \ell(y_{t+1}, u).$$

We will show that this lead to a contradiction.

We consider the optimal segmentation $(\mathbf{u}, \mathbf{t}) \in \mathcal{I}_{t+1}^K$ which achieves the optimum of $C_{K,t+1}(u)$. We consider two possible cases:

Scenario 1: $t_K < t$**.** Define $\mathbf{t}'$ such that for all $i < K$, we have $t'_i = t_i$ and $t'_K = t$. We have $(\mathbf{u}, \mathbf{t}') \in \mathcal{I}_t^K$. We can thus decompose $C_{K,t+1}(u)$ as

$$C_{K,t+1}(u) = \sum_{k=1}^{K} h_{t'_{k-1},t'_k}(u_k) + \ell(y_{t+1}, u).$$

By assumption we would recover $\sum_{k=1}^{K} h_{t'_{k-1},t'_k}(u_k) < C_{K,t}(u)$ which is a contradiction by definition of $C_{K,t}(u)$.

Scenario 2: $t_K = t$**.** Define $\mathbf{t}'$ such that for all $i < K-1$, we have $t'_i = t_i$ and $t'_{K-1} = t$. Also define $\mathbf{u}'$ such that for all $k \leq K-1$, we have $u'_k = u_k$. Thus $(\mathbf{u}', \mathbf{t}') \in \mathcal{I}_t^{K-1}$, and can then decompose $C_{K,t+1}(u)$ as

$$C_{K,t+1}(u) = \sum_{k=1}^{K} h_{t'_{k-1},t'_k}(u'_k) + \ell(y_{t+1}, u).$$

By assumption we would recover $\sum_{k=1}^{K-1} h_{t'_{k-1},t'_k}(u'_k) < C_{K-1,t}(u)$ which is a contradiction by definition of $C_{K-1,t}(u)$.

□

We have thus proved that the dynamic programming update rules can be used for computing the optimal cost functions $C_{k,t}$.

B Algorithm pseudocode

In this section we give pseudocode for our proposed Generalized Pruned Dynamic Programming Algorithm (GPDPA), and related algorithms.

B.1 GPDPA for reduced isotonic regression

We begin by providing a pseudocode solver for the simplest case, the reduced isotonic regression problem. We propose the following data structures and sub-routines for the computation:

- FunctionPiece: a data structure which represents one piece of a $C_{k,t}(u)$ cost function (for one interval of mean values u). It has coefficients which depend on the convex loss function ℓ (for the square loss it has three real-valued coefficients a, b, c which define a function au^2+bu+c). It also has two real-valued elements for min/max mean values $[\underline{u}, \overline{u}]$ of this interval, meaning the function $C_{k,t}(u) = au^2 + bu + c$ for all $u \in [\underline{u}, \overline{u}]$. Finally it stores a previous segment endpoint t' (integer) and mean u' (real).
- FunctionPieceList: an ordered list of FunctionPiece objects, which exactly stores a cost function $C_{k,t}(u)$ for all values of last segment mean u.
- OnePiece$(y, \underline{u}, \overline{u})$: a sub-routine that initializes a FunctionPieceList with just one FunctionPiece $\ell(y, u)$ defined on $[\underline{u}, \overline{u}]$.
- MinLess(t, f): an algorithm that inputs a changepoint and a FunctionPieceList, and outputs the corresponding min-less operator $f^{\leq}$ (another FunctionPieceList), with the previous changepoint set to $t' = t$ for each of its pieces. This algorithm also needs to store the previous mean value u' for each of the function pieces (see pseudocode below).
- MinOfTwo(f_1, f_2): an algorithm that inputs two FunctionPieceList objects, and outputs another FunctionPieceList object which is their minimum.
- ArgMin(f): an algorithm that inputs a FunctionPieceList and outputs three values: the optimal mean $u^* = \arg\min_u f(u)$, the previous segment end t' and mean u'.
- FindMean(u, f) an algorithm that inputs a mean value and a FunctionPieceList. It finds the FunctionPiece in f with mean $u \in [\underline{u}, \overline{u}]$ contained in its interval, then outputs the previous segment end t' and mean u' stored in that FunctionPiece.

The above data structures and sub-routines are used in the following pseudocode, which describes the GPDPA for solving the reduced isotonic regression problem.

Algorithm 1 Generalized Pruned Dynamic Programming Algorithm (GPDPA) for solving the reduced isotonic regression problem.

Input: data set $\mathbf{y} \in \mathbb{R}^n$, maximum number of segments $K \in \{2, \dots, n\}$.
Output: matrices of optimal segment means $U \in \mathbb{R}^{K \times K}$ and ends $T \in \{1, \dots, n\}^{K \times K}$
Compute min $\underline{y}$ and max $\overline{y}$ of $\mathbf{y}$.
$C_{1,1} \leftarrow$ OnePiece$(y_1, \underline{y}, \overline{y})$
for data points t from 2 to n:
    $C_{1,t} \leftarrow$ OnePiece$(y_t, \underline{y}, \overline{y}) + C_{1,t-1}$
for segments k from 2 to K: for data points t from k to n: // dynamic programming
    min_prev $\leftarrow$ MinLess$(t-1, C_{k-1,t-1})$ // this is $C^{\leq}_{k-1,t-1}$
    min_new $\leftarrow$ min_prev if $t = k$, else MinOfTwo(min_prev, $C_{k,t-1}$)
    $C_{k,t} \leftarrow$ min_new + OnePiece$(y_t, \underline{y}, \overline{y})$
for segments k from 1 to K: // decoding for every model size k
    $u^*, t', u' \leftarrow$ ArgMin$(C_{k,n})$
    $U_{k,k} \leftarrow u^*$; $T_{k,k} \leftarrow t'$ // store mean of segment k and end of segment $k-1$
    for segment s from $k-1$ to 1: // decoding for every segment $s < k$
        if $u' < \infty$: $u^* \leftarrow u'$ // equality constraint active, $u_s = u_{s+1}$
        $t', u' \leftarrow$ FindMean$(u^*, C_{s,t'})$
        $U_{k,s} \leftarrow u^*$; $T_{k,s} \leftarrow t'$ // store mean of segment s and end of segment $s-1$

Algorithm 1 begins by computing the min/max on line 3. The main storage of the algorithm is $C_{k,t}$, which should be initialized as a $K \times n$ array of empty FunctionPieceList objects. The computation of $C_{1,t}$ for all t occurs on lines 4–6.

The dynamic programming updates occur in the for loops on lines 7–10. Line 8 uses the MinLess sub-routine to compute the temporary FunctionPieceList min_prev (which represents the function $C^{\leq}_{k-1,t-1}$). Line 9 sets the temporary FunctionPieceList min_new to the cost of the only possible changepoint if $t = k$; otherwise, it uses the MinOfTwo sub-routine to compute the cost of the best changepoint for every possible mean value. Line 10 adds the cost of data point t, and stores the resulting FunctionPieceList in $C_{k,t}$.

The decoding of the optimal segment mean U (a $K \times K$ array of real numbers) and end T (a $K \times K$ array of integers) variables occurs in the for loops on lines 11–17. For a given model size k, the decoding begins on line 12 by using the ArgMin sub-routine to solve $u^* = \arg\min_u C_{k,n}(u)$ (the optimal values for the previous segment end t' and mean u' are also returned). Now we know that u^* is the optimal mean of the last (k-th) segment, which occurs from data point $t' + 1$ to n. These values are stored in $U_{k,k}$ and $T_{k,k}$ (line 13). And we already know that the optimal mean of segment $k - 1$ is u'. Note that the $u' = \infty$ flag means that the equality constraint is active (line 15). The decoding of the other segments $s < k$ proceeds using the FindMean sub-routine (line 16). It takes the cost $C_{s,t'}$ of the best model in s segments up to data point t', finds the FunctionPiece that stores the cost of u^*, and returns the new optimal values of the previous segment end t' and mean u'. The mean of segment s is stored in $U_{k,s}$ and the end of segment $s - 1$ is stored in $T_{k,s}$ (line 17).

The time complexity of Algorithm 1 is $O(KnI)$ where I is the complexity of the MinLess and MinOfTwo sub-routines, which is linear in the number of intervals (FunctionPiece objects) that are used to represent the cost functions. There are pathological synthetic data sets for which the number of intervals $I = O(n)$, implying a time complexity of $O(Kn^2)$. However, the average number of intervals in real data sets is empirically $I = O(\log n)$, so the average time complexity of Algorithm 1 is $O(Kn \log n)$.

B.2 MinLess algorithm

The MinLess algorithm implements the min-less operator $f^{\leq}$ (Definition 3), which is an essential sub-routine of the GPDPA. The following sub-routines are used to implement the MinLess algorithm.

- GetCost(p, u): an algorithm that takes a FunctionPiece object p, and a mean value u, and computes the cost at u. For a square loss FunctionPiece p with coefficients $a, b, c \in \mathbb{R}$, we have GetCost$(p, u) = au^2 + bu + c$.
- OptimalMean(p): an algorithm that takes one FunctionPiece object, and computes the optimal mean value. For a square loss FunctionPiece p we have OptimalMean$(p) = -b/(2a)$.
- ComputeRoots(p, d): an algorithm that takes one FunctionPiece object, and computes the solutions to $p(u) = d$. For the square loss we propose to use the quadratic formula. For other convex losses that do not have closed form expressions for their roots, we propose to use Newton's root finding method. Note that for some constants d there are no roots, and the algorithm needs to report that.
- f.push_piece$(\underline{u}, \overline{u}, p, u')$: push a new FunctionPiece at the end of FunctionPieceList f, with coefficients defined by FunctionPiece p, on interval $[\underline{u}, \overline{u}]$, with previous segment mean set to u'.
- ConstPiece(c): sub-routine that initializes a FunctionPiece p with constant cost c (for the square loss it sets $a = b = 0$ in $au^2 + bu + c$).

Algorithm 2 MinLess algorithm.

```
1: Input: The previous segment end t_prev (an integer), and f_in (a FunctionPieceList).
2: Output: FunctionPieceList f_out, initialized as an empty list.
3: prev_cost ← ∞
4: new_lower_limit ← LowerLimit(f_in[0]).
5: i ← 0; // start at FunctionPiece on the left
6: while i < Length(f_in): // continue until FunctionPiece on the right
7:    FunctionPiece p ← f_in[i]
8:    if prev_cost = ∞: // look for min in this interval.
9:       candidate_mean ← OptimalMean(p)
10:      if LowerLimit(p) < candidate_mean < UpperLimit(p):
11:         new_upper_limit ← candidate_mean // Minimum found in this interval.
12:         prev_cost ← GetCost(p, candidate_mean)
13:         prev_mean ← candidate_mean
14:      else: // No minimum in this interval.
15:         new_upper_limit ← UpperLimit(p)
16:      f_out.push_piece(new_lower_limit, new_upper_limit, p, ∞)
17:      new_lower_limit ← new_upper_limit
18:      i ← i + 1
19:   else: // look for equality of p and prev_cost
20:      (small_root, large_root) ← ComputeRoots(p, prev_cost)
21:      if LowerLimit(p) < small_root < UpperLimit(p):
22:         f_out.push_piece(new_lower_limit, small_root, ConstPiece(prev_cost), prev_mean)
23:         new_lower_limit ← small_root
24:         prev_cost ← ∞
25:      else: // no equality in this interval
26:         i ← i + 1 // continue to next FunctionPiece
27: if prev_cost < ∞: // ending on constant piece
28:    f_out.push_piece(new_lower_limit, UpperLimit(p), ConstPiece(prev_cost), prev_mean)
29: Set all previous segment end t′ = t_prev for all FunctionPieces in f_out
```

Consider Algorithm 2 which contains pseudocode for the computation of the min-less operator. The algorithm initializes prev_cost (line 3), which is a state variable that is used on line 8 to decide whether the algorithm should look for a local minimum or an intersection with a finite cost. Since prev_cost is initially set to ∞, the algorithm begins by following the convex function pieces from left to right until finding a local minimum. If no minimum is found in a given convex FunctionPiece (line 15), it is simply pushed on to the end of the new FunctionPieceList (line 16). If a minimum occurs within an interval (line 10), the cost and mean are stored (lines 11–12), and a new convex FunctionPiece is created with upper limit ending at that mean value (line 16). Then the algorithm starts looking for another FunctionPiece with the same cost, by computing the smaller root of the convex loss function (line 20). When a FunctionPiece is found with a root in the interval (line 21), a new constant FunctionPiece is pushed (line 22), and the algorithm resumes searching for a minimum. At the end of the algorithm, a constant FunctionPiece is pushed if necessary (line 28). The complexity of this algorithm is $O(I)$ where I is the number of FunctionPiece objects in f_{in}.

The algorithm which implements the min-more operator is analogous. Rather than searching from left to right, it searches from right to left. Rather than using the small root (line 21), it uses the large root.

B.3 Implementation details

Some implementation details that we found to be important:

Weights for data sequences that contain repeats it is computationally advantageous to use a run-length encoding of the data, and a corresponding loss function. For example if the data sequence 5,1,1,1,0,0,5,5 is encoded as

	Segment Neighborhood	Optimal Partitioning
unconstrained	PDPA	FPOP
constrained	GPDPA	GFPOP

Table 2: Algorithms for solving constrained and unconstrained versions of the Segment Neighborhood and Optimal Partitioning problems. PDPA = Pruned Dynamic Programming Algorithm, FPOP = Functional Pruning Optimal Partitioning, G = Generalized (can handle affine constraints on adjacent segment means).

$n = 4$ counts y_t 5,1,0,5 with corresponding weights w_t 1,3,2,2 then the Poisson loss function for mean μ is $\ell(y_t, w_t, \mu) = w_t(\mu - y_t \log \mu)$.

Mean cost The text defines $C_{k,t}$ functions as the total cost. However for very large data sets the cost values will be very large, resulting in numerical instability. To overcome this issue we instead implemented update rules using the mean cost. For weights $W_t = \sum_{i=1}^{t} w_i$, the update rule to compute the mean cost is

$$C_{k,t}(\mu) = \frac{1}{W_t}\left[\ell(y_t, \mu) + W_{t-1} \min\{C_{k,t-1}(\mu),\, C^{\leq}_{k-1,t-1}(\mu)\}\right]$$

Intervals in log(mean) space For the Poisson model of non-negative count data $y_t \in \{0, 1, 2, \dots\}$ there is no possible mean μ value less than 0. We thus used $\log(\mu)$ values to implement intervals in FunctionPiece objects. For example rather than storing $\mu \in [0, 1]$ we store $\log \mu \in [-\infty, 0]$.

Root finding For the ComputeRoots sub-routine for the Poisson loss, we used Newton root finding. For the larger root we solve $a \log \mu + b\mu + c = 0$ (linear as $\mu \to \infty$) and for the smaller root we solve $ax + be^x + c = 0$ ($x = \log \mu$, linear as $x \to -\infty$ and $\mu \to 0$). We stop the root finding when the cost is near zero (absolute cost value less than 10^{-12}).

Storage Since the dynamic programming update rule for $C_{k,t}$ only depends on $C^{\geq}_{k-1,t-1}$ and $C_{k,t-1}$, these are the only functions that need to be in memory, and the rest of the cost functions can be stored on disk (until the decoding step). We used the Berkeley DB Standard Template Library to store all the $C_{k,t}$ as a vector of FunctionPieceList objects.

B.4 Penalized version of reduced isotonic regression

Maidstone et al. [2016] proposed the Functional Pruning Optimal Partitioning (FPOP) algorithm to solve the "penalized" or "optimal partitioning" version of the segment neighborhood problem, where the constraint of K segments is replaced by a non-negative penalty $\lambda \in \mathbb{R}_+$ on the number of changes in the objective function. Rather than computing all models from 1 to K segments (as in the PDPA), the FPOP algorithm computes the single model with K segments (without computing models from 1 to $K - 1$ segments). The same penalization idea can be applied to models with affine constraints between adjacent segment means. The penalized version of the reduced isotonic regression problem (3) can be stated as

$$\begin{aligned} \underset{\substack{\mathbf{m}\in\mathbb{R}^n \\ \mathbf{c}\in\{0,1\}^{n-1}}}{\text{minimize}} \quad & \sum_{t=1}^{n} \ell(y_t, m_t) + \lambda \sum_{t=1}^{n-1} I(c_t \neq 0) \\ \text{subject to} \quad & c_t = 0 \Rightarrow m_t = m_{t+1} \\ & c_t = 1 \Rightarrow m_t \leq m_{t+1}. \end{aligned} \tag{20}$$

Note that the c_t variable is a changepoint indicator. The same functional pruning techniques used for the GPDPA can be exploited to create a solver for this problem. This results in the Generalized Functional Pruning Optimal Partitioning Algorithm (GFPOP, see Table 2).

Let $\overline{C}_{\lambda,t}(u)$ be the penalized cost of the most likely segmentation up to data point t, with last segment mean u. The initialization for the first data point is $\overline{C}_{\lambda,1}(u) = \ell(y_1, u)$. The dynamic programming update rule for all data points $t > 1$ is

$$\overline{C}_{\lambda,t}(u) = \ell(y_t, u) + \min\{\overline{C}^{\leq}_{\lambda,t-1}(u) + \lambda,\ \overline{C}_{\lambda,t-1}(u)\}. \tag{21}$$

The same sub-routines described in Section B.2 can be used to implement the algorithm below, which solves the penalized reduced isotonic regression problem (20).

Algorithm 3 Generalized Functional Pruning Optimal Partitioning (GFPOP) for penalized reduced isotonic regression

1: Input: data set $\mathbf{y} \in \mathbb{R}^n$, penalty constant $\lambda \geq 0$.
2: Output: vectors of optimal segment means $U \in \mathbb{R}^n$ and ends $T \in \{1, \ldots, n\}^n$
3: Compute min $\underline{y}$ and max $\overline{y}$ of $\mathbf{y}$.
4: $\overline{C}_{\lambda,1} \leftarrow$ OnePiece$(y_1, \underline{y}, \overline{y})$
5: for data points t from 2 to n: // dynamic programming
6: min_prev $\leftarrow \lambda +$ MinLess$(t-1, \overline{C}_{\lambda,t-1})$
7: min_new $\leftarrow$ MinOfTwo(min_prev, $\overline{C}_{\lambda,t-1}$)
8: $\overline{C}_{\lambda,t} \leftarrow$ min_new + OnePiece$(y_t, \underline{y}, \overline{y})$
9: $u^*, t', u' \leftarrow$ ArgMin$(\overline{C}_{\lambda,n})$ // begin decoding
10: $i \leftarrow 1$; $U_i \leftarrow u^*$; $T_i \leftarrow t'$
11: while $t' > 0$:
12: if $u' < \infty$: $u^* \leftarrow u'$
13: $t', u' \leftarrow$ FindMean$(u^*, \overline{C}_{\lambda,t'})$
14: $i \leftarrow i + 1$; $U_i \leftarrow u^*$; $T_i \leftarrow t'$

Algorithm 3 begins by computing the min/max (line 3). The main storage of the algorithm is $\overline{C}_{\lambda,t}$, which should be initialized as an array of n empty FunctionPieceList objects.

The dynamic programming recursion in this algorithm has only one for loop over data points t (line 5). The penalty constant λ is added to all of the function pieces that result from MinLess (line 6), before computing MinOfTwo (line 7). The last step of each dynamic programming update is to add the cost of the new data point (line 8).

The decoding process on lines 9–14 is essentially the same as the GPDPA (Algorithm 1). The last segment mean and second to last segment end are first stored on line 10 in (U_1, T_1). For each other segment i, the mean and previous segment end are stored on line 14 in (U_i, T_i). Note that there should be space to store (U_i, T_i) parameters for up to n segments. However, there are usually less than n segments, and the algorithm should return a special flag for unused parameters, for example $(U_i = \infty, T_i = -1)$.

The time complexity of Algorithm 3 is $O(nI)$, where I is the time complexity of the MinLess and MinOfTwo sub-routines. As in the GPDPA, the time complexity of these sub-routines is linear in the number of intervals (FunctionPiece objects) that are used to represent the $\overline{C}_{\lambda,t}$ cost functions. Since the number of intervals in real data is typically $I = O(\log n)$ (see Section 5.1), the overall time complexity of Algorithm 3 is on average $O(n \log n)$.

B.5 Generalized Functional Pruning Optimal Partitioning Solvers

The GFPOP algorithm can solve problems with more general constraints than reduced isotonic regression. Let $G = (V, E)$ be a directed graph that represents the model constraints (for examples see Figure 5). The vertices $V = \{1, \ldots, |V|\}$ can be represented as integers, one for every distinct state. The edges $E = \{1, \ldots, |E|\}$ is another set of integers, each of which represents one of the possible changes between states. Each edge/change $c \in E$ has corresponding data $(\underline{v}_c, \overline{v}_c, \lambda_c, g_c)$ which specifies a transition from state $\underline{v}_c$ to state $\overline{v}_c$, with a penalty of $\lambda_c \in \mathbb{R}_+$, and a constraint function $g_c : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$.

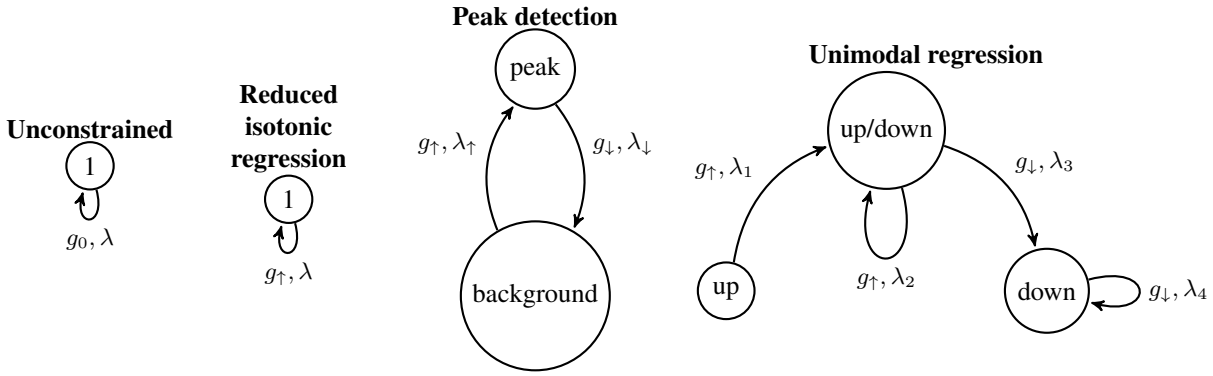

Figure 5: Examples of state graphs for four models. Nodes represent states and edges represent changes. Each change has a corresponding penalty λ, and a function g that determines what types of changes are possible (g_0 any change, $g_\uparrow$ non-decreasing, $g_\downarrow$ non-increasing). Even if there is no edge from a node to itself, it is still possible to stay in the same state without introducing a changepoint and penalty. Note that the Unimodal regression state X should be interpreted as "can change X" e.g. up/down means "can change up/down."

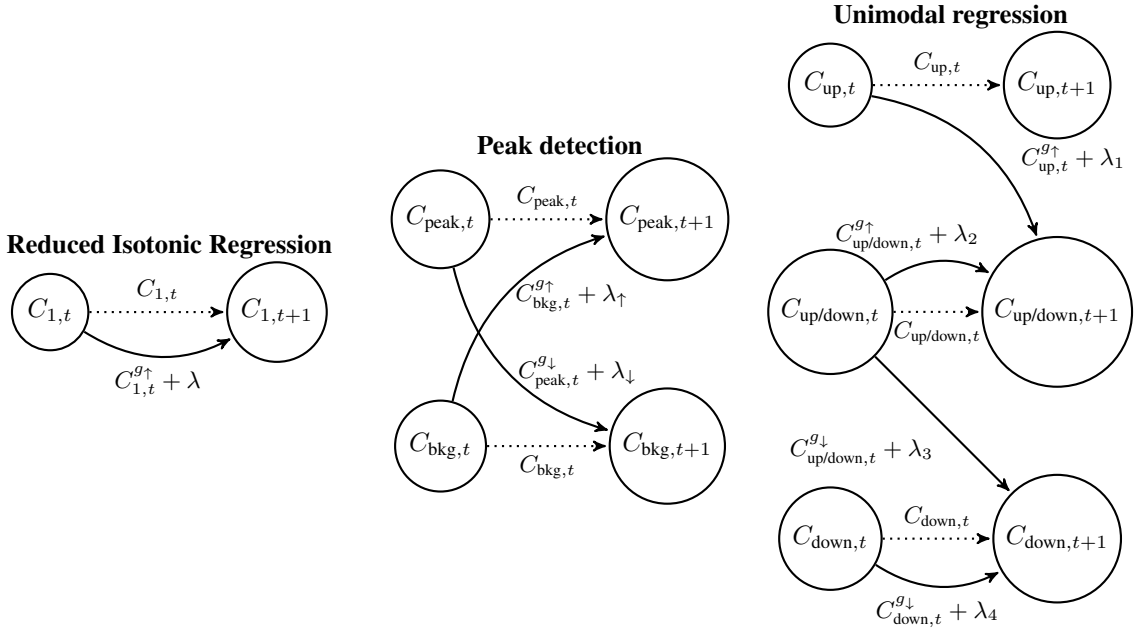

Figure 6: Computation graphs which represent the dynamic programming updates (23) for the state graph models in Figure 5. Nodes represent cost functions $C_{s,t}$ in each state s at data t and $t+1$. Edges represent inputs to the $\min\{\}$ operation (solid for a changepoint, dotted for no change). For example in reduced isotonic regression, $C_{1,t+1}(u) = \ell(y_{t+1}, u) + \min\{C_{1,t}(u), C^{g_\uparrow}_{1,t}(u) + \lambda\}$.

In the optimization problem below we also allow $c = 0$, which implies no penalty $\lambda_0 = 0$, and means no change:

$$\begin{aligned} &\underset{\substack{\mathbf{m}\in\mathbb{R}^n,\ \mathbf{s}\in V^n \\ \mathbf{c}\in\{0,1,\ldots,|E|\}^{n-1}}}{\text{minimize}} \quad \sum_{t=1}^{n} \ell(y_t, m_t) + \sum_{t=1}^{n-1} \lambda_{c_t} \qquad (22) \\ &\text{subject to} \quad c_t = 0 \Rightarrow m_t = m_{t+1} \text{ and } s_t = s_{t+1} \\ &\qquad\qquad\ \ c_t \neq 0 \Rightarrow g_{c_t}(m_t, m_{t+1}) \leq 0 \text{ and } (s_t, s_{t+1}) = (\underline{v}_{c_t}, \overline{v}_{c_t}). \end{aligned}$$

If some states are desired at the start or end, then those constraints $s_1 \in \underline{S}$, $s_n \in \overline{S}$ can also be enforced. To compute the solution to this optimization problem, we propose the following dynamic programming algorithm.

Let $C_{s,t}(u)$ be the optimal cost with mean u and state s at data point t. This quantity can be recursively computed using dynamic programming. The initialization for the first data point is $C_{s,1}(u) = \ell(y_1, u)$ for all states s. The dynamic programming update rule for all data points $t > 1$ is

$$C_{s,t}(u) = \ell(y_t, u) + \min\{M_{s,t-1}(u),\, C_{s,t-1}(u)\}, \tag{23}$$

where the minimum cost of all possible changes to state s from time point $t-1$ is

$$M_{s,t-1}(u) = \min_{c \in E_s} C^{g_c}_{\underline{v}_c,t-1}(u) + \lambda_c, \tag{24}$$

and the set of all changes going to state s is

$$E_s = \{c \in E \mid \overline{v}_c = s\}. \tag{25}$$

The computations required for the dynamic programming updates (23) can be visualized using a computation graph (Figure 6).

The pseudocode for the algorithm which implements the dynamic programming updates (23) is stated below.

Algorithm 4 Generalized Functional Pruning Optimal Partitioning Algorithm, Dynamic Programming (GFPOP-DP)

```
1: Input: data y and weights w (both size n), number of vertices/states |V|, starts S ⊆ V, edges/transitions E.
2: Allocate |V| × n array of optimal cost functions C_{s,t}, each initialized to NULL.
3: for t from 1 to n:
4:    if t == 1:
5:       for s in S: // initialize cost for all possible starting states
6:          C_{s,t} ← InitialCost(y_t, w_t)
7:    else:
8:       for s from 1 to |V|:
9:          if C_{s,t-1} is NOT NULL: // previous cost in this state has been computed
10:            C_{s,t} ← C_{s,t-1} // cost of staying in this state (no change)
11:      for (v, v̄, λ, ConstrainedCost) in E:
12:         if C_{v,t-1} is NOT NULL: // previous cost has been computed
13:            cost_of_change ← ConstrainedCost(C_{v,t-1})
14:            cost_of_change.set(v, t − 1)
15:            cost_of_change.addPenalty(λ)
16:            if C_{v̄,t} is NULL:
17:               C_{v̄,t} ← cost_of_change
18:            else:
19:               C_{v̄,t} ← MinOfTwo(C_{v̄,t}, cost_of_change)
20:      for s from 1 to |V|:
21:         if C_{s,t} is NOT NULL:
22:            C_{s,t}.addDataPoint(y_t, w_t)
23: Output: |V| × n array of optimal cost functions C_{s,t}.
```

The algorithm above performs several checks if $C_{s,t}$ is NULL or not (lines 9, 12, 16, 21). All costs are initialized as NULL (line 2). After having performed the cost update for data t, a NULL cost $C_{s,t}$ means that state s is not feasible at data t. For each constraint function g there is a corresponding ConstrainedCost sub-routine that is mentioned on lines 11 and 13 (e.g. no constraint g_0 MinUnconstrained, non-decreasing change $g_\uparrow$ MinLess, non-increasing change $g_\downarrow$ MinMore).

The average time and space complexity of Algorithm 4 is $O(|V|nI)$ where $|V|$ is the number of states and I is the average number of of intervals stored in the $|V| \times n$ array of $C_{s,t}$ cost functions. We observed that $I = \log n$

in the empirical tests of the peak detection model on ChIP-seq data (Section 5.1), so we expect that the average time complexity of Algorithm 4 is $O(|V|n \log n)$.

Note that the algorithm above only performs the dynamic programming. The decoding of optimal model parameters is achieved using the algorithm below.

Algorithm 5 Generalized Functional Pruning Optimal Partitioning Algorithm, decoding (GFPOP-decode)

Output: $|V| \times n$ array of optimal cost functions $C_{s,t}$, ends $\overline{S} \subseteq V$.
Allocate $\mathbf{m} \in \mathbb{R}^n$ (mean), $\mathbf{s} \in \mathbb{Z}^n$ (state), $\mathbf{t} \in \mathbb{Z}^n$ (segment end).
$u^*, s^*, t', s' \leftarrow \text{ArgMin}(C_{\cdot,n}, \overline{S})$ // begin decoding
$i \leftarrow 1$; $m_i \leftarrow u^*$; $s_i \leftarrow s^*$; $t_i \leftarrow n$
while $t' > 0$:
    $i \leftarrow i + 1$; $t_i \leftarrow t'$
    $u^*, s^*, t', s' \leftarrow \text{ArgMin}(C_{s',t'})$
    $m_i \leftarrow u^*$; $s_i \leftarrow s^*$
Output: $\mathbf{m}, \mathbf{s}, \mathbf{t}$.

References

H. Ashoor, A. Hérault, A. Kamoun, F. Radvanyi, V. Bajic, E. Barillot, and V. Boeva. HMCan: a method for detecting chromatin modifications in cancer samples using ChIP-seq data. *Bioinformatics*, 29(23):2979–2986, 2013.

I. Auger and C. Lawrence. Algorithms for the optimal identification of segment neighborhoods. *Bull Math Biol*, 51: 39–54, 1989.

T. Bailey, P. Krajewski, I. Ladunga, C. Lefebvre, Q. Li, T. Liu, P. Madrigal, C. Taslim, and J. Zhang. Practical guidelines for the comprehensive analysis of ChIP-seq data. *PLoS computational biology*, 9(11), 2013.

R. Bellman. On the approximation of curves by line segments using dynamic programming. *Commun. ACM*, 4(6): 284–, June 1961.

M. Best and N. Chakravarti. Active set algorithms for isotonic regression; a unifying framework. *Mathematical Programming*, 47(1):425–439, 1990.

N. Haiminen, A. Gionis, and K. Laasonen. Algorithms for unimodal segmentation with applications to unimodality detection. *Knowledge and Information Systems*, 14(1):39–57, 2008.

J. Hardwick and Q. Stout. Optimal reduced isotonic regression. *arXiv preprint arXiv:1412.2844*, 2014.

T. Hocking, G. Rigaill, and G. Bourque. PeakSeg: constrained optimal segmentation and supervised penalty learning for peak detection in count data. In *Proc. 32nd ICML*, pages 324–332, 2015.

T. Hocking, P. Goerner-Potvin, A. Morin, X. Shao, T. Pastinen, and G. Bourque. Optimizing chip-seq peak detectors using visual labels and supervised machine learning. *Bioinformatics*, 2016.

H. Hoefling. A path algorithm for the fused lasso signal approximator. *Journal of Computational and Graphical Statistics*, 19(4):984–1006, 2010.

B. Jackson, J. Scargle, D. Barnes, S. Arabhi, A. Alt, P. Gioumousis, E. Gwin, P. Sangtrakulcharoen, L. Tan, and T. Tsai. An algorithm for optimal partitioning of data on an interval. *IEEE Signal Process Lett*, 12:105–108, 2005.

N. Johnson. *Efficient models and algorithms for problems in genomics*. PhD thesis, Stanford, 2011. https://purl.stanford.edu/jq411pj0455.

N. Johnson. A Dynamic Programming Algorithm for the Fused Lasso and L0-Segmentation. *Journal of Computational and Graphical Statistics*, 22(2):246–260, 2013.

R. Killick, P. Fearnhead, and I. Eckley. Optimal detection of changepoints with a linear computational cost. *arXiv:1101.1438*, Jan. 2011.

R. Maidstone, T. Hocking, G. Rigaill, and P. Fearnhead. On optimal multiple changepoint algorithms for large data. *Statistics and Computing*, pages 1–15, 2016. ISSN 1573-1375.

P. Mair, K. Hornik, and J. de Leeuw. Isotone optimization in R: pool-adjacent-violators algorithm (PAVA) and active set methods. *Journal of statistical software*, 32(5):1–24, 2009.

A. Cleynen and E. Lebarbier. Segmentation of the Poisson and negative binomial rate models: a penalized estimator. *ESAIM: PS*, 18:750–769, 2014.

G. Rigaill. Pruned dynamic programming for optimal multiple change-point detection. arXiv:1004.0887, 2010.

G. Rigaill. A pruned dynamic programming algorithm to recover the best segmentations with 1 to kmax change-points. *Journal de la Société Française de la Statistique*, 156(4), 2015.

G. Rote. Isotonic regression by dynamic programming. http://www.inf.fu-berlin.de/lehre/WS12/HA/isotonicregression.pdf, unpublished.

M. Schell and B. Singh. The reduced monotonic regression method. *Journal of the American Statistical Association*, 92(437):128–135, 1997.

A. Scott and M. Knott. A cluster analysis method for grouping means in the analysis of variance. *Biometrics*, 30: 507–512, 1974.

R. Tibshirani, H. Hoefling, and R. Tibshirani. Nearly-isotonic regression. *Technometrics*, 53(1):54–61, 2011.

Y. Zhang, T. Liu, C. Meyer, J. Eeckhoute, D. Johnson, B. Bernstein, C. Nusbaum, R. Myers, M. Brown, W. Li, et al. Model-based analysis of ChIP-Seq (MACS). *Genome Biol*, 9(9):R137, 2008.