Journal of Machine Learning Research 21 (2020) 1-40 Submitted 12/18; Revised 1/20; Published 3/20

Constrained Dynamic Programming and Supervised Penalty Learning Algorithms for Peak Detection in Genomic Data

Toby Dylan Hocking TOBY.HOCKING@NAU.EDU
Northern Arizona University, School of Informatics, Computing, and Cyber Systems
1295 S. Knoles Dr., Building 90, Room 210, Flagstaff, AZ, 86011, USA

Guillem Rigaill GUILLEM.RIGAILL@INRAE.FR
Université Paris-Saclay, CNRS, Univ Evry, Laboratoire de Mathématiques et Modélisation d'Evry, 91037, Evry, France

Paul Fearnhead P.FEARNHEAD@LANCASTER.AC.UK
Lancaster University, Department of Mathematics and Statistics,
Fylde College, Lancaster, LA1-4YF, UK

Guillaume Bourque GUIL.BOURQUE@MCGILL.CA
McGill University, Department of Human Genetics
740, Dr Penfield Avenue, Montréal, Québec, H3A-0G1, Canada

Editor: Animashree Anandkumar

Abstract

Peak detection in genomic data involves segmenting counts of DNA sequence reads aligned to different locations of a chromosome. The goal is to detect peaks with higher counts, and filter out background noise with lower counts. Most existing algorithms for this problem are unsupervised heuristics tailored to patterns in specific data types. We propose a supervised framework for this problem, using optimal changepoint detection models with learned penalty functions. We propose the first dynamic programming algorithm that is guaranteed to compute the optimal solution to changepoint detection problems with constraints between adjacent segment mean parameters. Implementing this algorithm requires the choice of penalty parameter that determines the number of segments that are estimated. We show how the supervised learning ideas of Rigaill et al. (2013) can be used to choose this penalty. We compare the resulting implementation of our algorithm to several baselines in a benchmark of labeled ChIP-seq data sets with two different patterns (broad H3K36me3 data and sharp H3K4me3 data). Whereas baseline unsupervised methods only provide accurate peak detection for a single pattern, our supervised method achieves state-of-the-art accuracy in all data sets. The log-linear timings of our proposed dynamic programming algorithm make it scalable to the large genomic data sets that are now common. Our implementation is available in the PeakSegOptimal R package on CRAN.

Keywords: Non-convex, constrained, optimization, changepoint, segmentation.

1. Introduction

In recent years, high-throughput DNA sequencing technologies have been improving at a rapid pace, resulting in progressively bigger genomic data sets. Two common genome-wide assays are ChIP-seq, a genome-wide assay for histone modifications or transcription factor binding sites (Barski et al., 2007); and ATAC-seq, an Assay for Transposase-Accessible

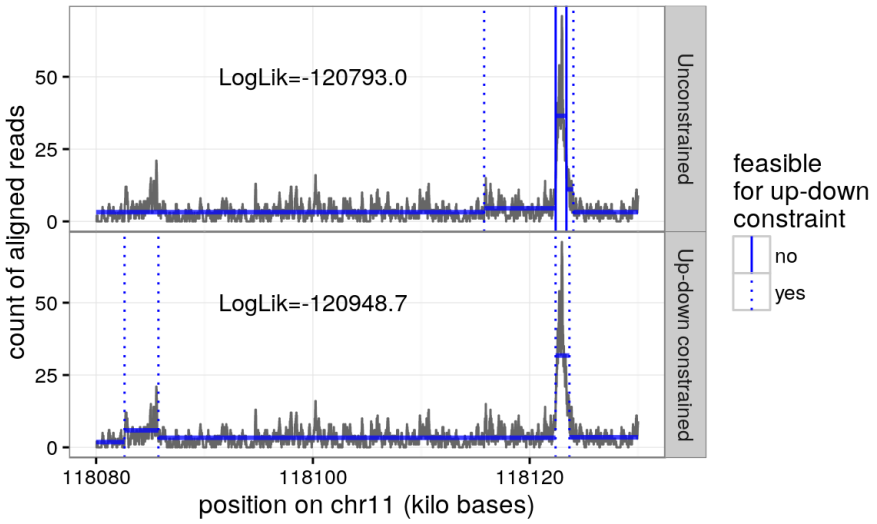


Figure 1: ChIP-seq read count data (grey lines) for one sample on a subset of chromosome 11. **Top:** the maximum likelihood Poisson model with 5 segment means (horizontal blue lines) has two up changes followed by two down changes (vertical blue lines). It does not satisfy the up-down constraint (odd-numbered changes must be up, and even-numbered changes must be down). **Bottom:** the up-down constrained model has a lower log-likelihood value, but each up change is followed by a down change. Odd-numbered segments are interpreted as background noise, and even-numbered segments are interpreted as peaks (a short peak on the left and a tall peak on the right).

Chromatin which measures open chromatin (Buenrostro et al., 2015). These experiments have been used to characterize the epigenome of samples in large-scale mapping projects such as ENCODE (ENCODE Project Consortium, 2011). Briefly, each assay yields a set of DNA sequence reads which are aligned to a reference genome, and then the number of aligned reads are counted at each genomic position (Figure 1). This results in a vector of non-negative integer counts $\mathbf{y} \in \mathbb{Z}_+^n$ over n positions.

The size n of these aligned read count data depends on the size of contiguous regions on chromosomes in the reference genome. For example, the largest chromosome in the human genome (hg19) is chr1, which has $n = 249,250,621$ bases (distinct positions at which the number of aligned reads is measured). Analysis of such data thus requires computationally efficient algorithms which scale to data sets of arbitrarily large size.

Although these read counts can be interpreted as quantitative data, they are most often interpreted using one of the many available peak detection algorithms (Wilibanks and Facciotti, 2010; Rye et al., 2010; Szalkowski and Schmid, 2011). A peak detection algorithm is a binary classifier $c(\mathbf{y}) \in \{0,1\}^n$ for each genomic position, where peaks are the 1-class, and background noise is the 0-class. Importantly, peaks and background occur in long contiguous segments across the genome. Typical algorithms from the bioinformatics literature such as MACS (Zhang et al., 2008) and HMCan (Ashoor et al., 2013) are unsupervised heuristics with several parameters that affect peak detection accuracy (window/bin sizes, p-value

thresholds, etc). Although such algorithms are fast for large data sets, they are typically accurate only for a specific data/pattern type (e.g. MACS works for sharp H3K4me3 but not broad H3K36me3 data) (Hocking et al., 2016).

Hidden Markov Models (HMMs) with common mean parameters for the peak or background regions could be used to model such sequence data, but we do not explore them in this paper for two reasons. First, we have observed in real ChIP-seq data that background and peak means are not constant throughout the genome (Supplementary Figure 1), so shared mean parameters would not be a good fit. Second, inference algorithms for HMMs are only guaranteed to find a local maximum of the likelihood; we are more interested in changepoint detection models with dynamic programming algorithms that can provably compute the global maximum of the corresponding likelihood. Such optimal inference algorithms only work for models with separate parameters for each segment.

Recently Cleynen and Lebarbier (2014) proposed a Pruned Dynamic Programming Algorithm (PDPA) for computing the most likely K segment means (and $K-1$ changepoints) using a Poisson model. This is computed for a range of segments K, which acts as a regularization parameter. Small values of K result in too few segments/peaks (false negative peak detections, underfitting), and large values of K result in too many (false positive peak detections, overfitting). Oracle penalties (Cleynen and Lebarbier, 2014) or learned penalties (Rigaill et al., 2013) can be used to select the number of segments K. Because this model sometimes has several consecutive up changes, it is non-trivial to interpret in terms of peaks and background (Figure 1, top).

To ensure that the segmentation model is interpretable in terms of peaks and background, Hocking et al. (2015) introduced a Constrained Dynamic Programming Algorithm (CDPA) for computing a model where up changes are followed by down changes, and vice versa (Figure 1, bottom). The model with $P \in \{0, 1, \dots, \}$ peaks has $K = 2P + 1 \in \{1, 3, \dots\}$ segments. These constraints ensure that odd-numbered segments can be interpreted as background, and even-numbered segments can be interpreted as peaks. Without the up-down constraints, it becomes harder to identify which segments should be flagged as peak versus background—one would need some ad-hoc post-processing rules (see Section 5.2 for our analysis of three such rules). In a recent comparison study, Hocking et al. (2016) showed that this algorithm achieves state-of-the-art peak detection accuracy in a benchmark of ChIP-seq data sets which include both broad H3K36me3 and sharp H3K4me3 data/patterns. This constrained model is further justified by the statistical arguments of Gao et al. (2017), who show that using shape constraints can reduce the minimal risk bounds; with these being $O(\log \log n)$ for estimating changes in mean under a monotone constraint as compared to $O(\log n)$ without the constraint. The fact that using shape constraints can lead to a lower risk bound suggests that the constrained problem is statistically simpler than the unconstrained problem, and that it is worthwhile to integrate those constraints in the model if they are known to be true.

The previously proposed CDPA suffers from two major issues, which we fix in this paper using a rigorous mathematical analysis of the constrained maximum likelihood changepoint detection problem. First, the CDPA does not necessarily compute the globally optimal model, because it does not accurately account for the constraints (for a detailed explanation, see Supplementary Figure 2). Second, the time complexity of analyzing n data points with the CDPA is $O(Kn^2)$. Since this is quadratic in the number of data points, it can be too

Constraint	No pruning	Functional pruning
None	Dynamic Prog. Algo. (DPA) Optimal, $O(Kn^2)$ time Auger and Lawrence (1989)	Pruned DPA (PDPA) Optimal, $O(Kn \log n)$ time Rigaill (2010); Johnson (2011)
Up-down	Constrained DPA (CDPA) Sub-optimal, $O(Kn^2)$ time Hocking et al. (2015)	Generalized Pruned DPA (GPDPA) Optimal, $O(Kn \log n)$ time **This paper**

Table 1: Our main 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. For each algorithm we show whether or not it is guaranteed to compute the global optimum (optimal/sub-optimal), and its time complexity on average in our empirical tests on real ChIP-seq data sets.

slow for use on large genomic data sets which are now common. In this paper we propose a new algorithm that resolves both of these issues (Table 1).

1.1. Contributions and Organization

In Section 2 we define the optimal changepoint detection problems. The first problem is where there are no constraints between the segment-specific mean parameters. The second is where we impose the up-down constraint between the means of neighboring segments. Whilst several efficient dynamic programming algorithms can solve the unconstrained minimization, we are unaware of equivalent dynamic programming algorithms for exactly solving the constrained version. Our main contribution is described in Section 3, where we generalize the functional pruning technique of Rigaill (2010, 2015) so that it can be applied to the constrained minimization problem (Table 1), leading to the fast and optimal Generalized Pruned Dynamic Programming Algorithm (GPDPA). These new ideas around how to deal with constraints on the means for neighbouring segments is of independent interest. For example, since the first version of this paper appeared (Hocking et al., 2017), its ideas have already been used to model monotone constraints in spike detection problems for calcium imaging data (Jewell et al., 2019). In Section 4 we describe the labeled data sets and we propose a supervised learning algorithm for predicting the crucial penalty parameter, which determines the number of segments K. In Section 5 we show that our proposed algorithms achieve state-of-the-art speed and peak detection accuracy in a benchmark of ChIP-seq data sets. The paper ends with a discussion.

2. Unconstrained and Constrained Changepoint Models

In this section we define two related changepoint detection problems. We use tilde $\tilde{C}$ to denote the optimal unconstrained cost, and no tilde C to denote the up-down constrained cost. We use the notation $C^*_{K,n}, \tilde{C}^*_{K,n}$, where the star (*) indicates an optimal cost value

(a real number) and the subscripts refer to the number of segments K and the number of data points n.

2.1. Unconstrained changepoint model with K segments

The data consist of n observations denoted by the random vector $\mathbf{y} = (y_1, \ldots, y_n)$. We assume a piecewise constant mean model with $K-1$ changes (K distinct values). For each segment $k \in \{1 \ldots, K\}$, the real-valued mean parameter u_k is assigned to data points $(t_{k-1}, t_k]$. The first index, $t_0 = 0$, and last index, $t_K = n$, are fixed; the others, $t_1 < \cdots < t_{K-1}$, are changepoint variables.

For real-valued data $y_t \in \mathbb{R}$, the statistical model is for every segment $k \in \{1 \ldots, K\}$, each data point on that segment $t \in (t_{k-1}, t_k]$ is $y_t \overset{\text{iid}}{\sim} N(u_k, \sigma^2)$. Maximizing the likelihood in this Normal model is equivalent to minimizing the square loss $\ell(y_t, u_k) = (u_k - y_t)^2$. In the case of genomic count data, we have a sequence of non-negative integers $y_t \in \mathbb{Z}_+$, so we assume $y_t \overset{\text{iid}}{\sim} \text{Poisson}(u_k)$. Maximizing the likelihood in the Poisson model is equivalent to minimizing the Poisson loss $\ell(y_t, u_k) = u_k - y_t \log u_k$. In either case we can write the cost function for segment k as

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

The optimal cost in K segments up to n data points is defined as the solution of the optimization problem

$$\tilde{C}^*_{K,n} = \min_{\substack{u_1,\ldots,u_K \in \mathbb{R} \\ 0=t_0<t_1<\cdots<t_{K-1}<t_K=n}} \sum_{k=1}^{K} h_{t_{k-1},t_k}(u_k). \tag{2}$$

This optimization problem is non-convex because the changepoint variables t_k are integers. Nonetheless, the optimal solution can be computed in $O(Kn^2)$ time using a dynamic programming algorithm (Auger and Lawrence, 1989). By exploiting the structure of the loss function ℓ, the pruned dynamic programming algorithm of Rigaill (2010) computes the same optimal solution much faster; empirically having a computational cost that is $O(Kn \log n)$. Efficient algorithms exist for solving similar problems such as Optimal Partitioning (Jackson et al., 2005; Johnson, 2011; Killick et al., 2012; Maidstone et al., 2017) and the Fused Lasso (Hoefling, 2010).

2.2. Up-down constrained changepoint model with K segments

The peak detection problem we are interested in can be formulated in a similar way, but with the addition of further constraints on the segment means. These constraints force the mean to alternate between increasing and decreasing at each changepoint. Formally, the optimal cost in K segments up to n data points, subject to the up-down constraint, is

defined as

$$C^*_{K,n} = \min_{\substack{u_1,\ldots,u_K\in\mathbb{R} \\ 0=t_0<t_1<\cdots<t_{K-1}<t_K=n}} \sum_{k=1}^{K} h_{t_{k-1},t_k}(u_k) \quad (3)$$
$$\text{subject to} \quad u_{k-1} \leq u_k \ \forall k \in \{2,4,\ldots\},$$
$$u_{k-1} \geq u_k \ \forall k \in \{3,5,\ldots\}.$$

Note that the constraints between adjacent segment means are defined in terms of non-strict inequalities (e.g. $u_{k-1} \leq u_k$) so that the optimal means $u_1,\ldots,u_K$ and cost $C^*_{K,n}$ are always well-defined real numbers. In contrast, if strict equalities were used (e.g. $u_{k-1} < u_k$) the resulting problem could be unbounded from below; see Section 3.9 for an example. It also is worth noting the connection between this up-down constrained changepoint problem (3) and several related problems that have been previously studied. The *reduced* isotonic regression problem is similar (Schell and Singh, 1997), but uses only non-decreasing $u_{k-1} \leq u_k$ change constraints for all $k \in \{2,3,\ldots\}$. Hardwick and Stout (2014) proposed an efficient algorithm for reduced isotonic regression. Note that the well-known Pool-Adjacent-Violators Algorithm (PAVA) (Mair et al., 2009) solves a simpler problem (isotonic regression), without the constraint on the number of segments K. As far as we know, the PAVA has not been adapted for use with more complex constraints such as the up-down constraints we consider in this paper.

Our contribution in this paper is proving that the functional pruning technique of Rigaill (2010) and Maidstone et al. (2017) can be generalized to constrained changepoint models such as (3). The resulting Generalized Pruned Dynamic Programming Algorithm (GPDPA) computes the optimal solution to the up-down constrained changepoint model, and enjoys $O(Kn\log n)$ time complexity (on average in our empirical tests of real ChIP-seq data sets).

3. Dynamic Programming for Constrained Changepoint Models

In this section we first discuss unconstrained and constrained algorithms that use the classical dynamic recursion in terms of optimal cost values. We then present the new approach which recursively computes optimal cost functions.

3.1. Classical Dynamic Programming Approach

The classical dynamic programming algorithm of Auger and Lawrence (1989) recursively computes the optimal cost in K segments up to n data points via

$$\tilde{C}^*_{K,n} = \min_{t_{K-1}} \underbrace{\min_{\substack{u_1,\ldots,u_{K-1} \\ t_1<\cdots<t_{K-2}}} \sum_{k=1}^{K-1} h_{t_{k-1},t_k}(u_k)}_{\tilde{C}^*_{K-1,t_{K-1}}} + \underbrace{\min_{u_K} h_{t_{K-1},n}(u_K)}_{h^*_{t_{K-1},n}}. \quad (4)$$

The main idea of the classical dynamic programming recursion (4) is to separate the cost into two terms. The left term $\tilde{C}^*_{K-1,t_{K-1}}$ is the previously computed cost of the best model in $K-1$ segments up to data point t_{K-1}. The right term $h^*_{t_{K-1},n}$ is the optimal cost of the

K-th segment, which starts after the last changepoint t_{K-1} and continues until the final data point n. The minimization in (4) over all possible last changepoints $t_{K-1} \in \{K-1, \ldots, n-1\}$ can be computed in $O(n)$ time. The vector of $h^*_{K-1,n}, \ldots, h^*_{n-1,n}$ values can be computed in $O(n)$ time using the cumulative sum; computing the entire vector of $\tilde{C}^*_{K,K}, \ldots, \tilde{C}^*_{K,n}$ is therefore $O(n^2)$ using (4).

3.2. Classical Algorithm Modified for Constraints

In this section we review the Constrained Dynamic Programming Algorithm (CDPA), which we previously proposed (Hocking et al., 2015). In the CDPA we proposed to constrain the set of possible last changepoints t_{K-1} used in the minimization (4). In the right side of (4) the min with respect to the last segment mean u_K is achieved by

$$\hat{u}_{t_{K-1},n} = \arg\min_{u_K \in \mathbb{R}} h_{t_{K-1},n}(u_K), \tag{5}$$

which is the best mean value for the last segment starting after data point t_{K-1}. The main idea of the CDPA is to only consider last changepoints t_{K-1} which result in a last segment mean $\hat{u}_{t_{K-1},n}$ that jumps in the correct direction (up or down). More precisely, letting $\mathcal{U}_{K-1,\tau}$ be the best mean of the $K-1$-th segment up to data point τ, we consider the reduced set of possible changepoints

$$\mathcal{I}_{K,n} = \begin{cases} \tau \in \{K-1, \ldots, n-1\} \mid \mathcal{U}_{K-1,\tau} < \hat{u}_{\tau,n} & \text{if } K \text{ is even,} \\ \tau \in \{K-1, \ldots, n-1\} \mid \mathcal{U}_{K-1,\tau} > \hat{u}_{\tau,n} & \text{if } K \text{ is odd.} \end{cases} \tag{6}$$

That yields the recursive update rule for the constrained cost

$$c^*_{K,n} \quad = \quad \min_{\tau \in \mathcal{I}_{K,n}} c^*_{K-1,\tau} + h^*_{\tau,n}. \tag{7}$$

Here we use the lowercase c^* notation for the cost to emphasize that this update rule is a greedy heuristic; it does not always compute the global optimum $C^*_{K,n}$ of the constrained problem (3). Using this update rule, computing the entire matrix of constrained cost values $c^*_{k,t}$ for all $k \in \{1, \ldots, K\}$ and $t \in \{1, \ldots, n\}$ takes $O(Kn^2)$ time, which is slow and sub-optimal. We therefore propose a new update rule in the next section which is both faster and optimal.

3.3. Dynamic Programming with Functional Pruning

The key insight of functional pruning (Rigaill, 2010; Maidstone et al., 2017) is that the order of minimization can be reversed in (4). By taking out the minimization with respect to the last segment mean u_K, we obtain the following equation for the optimal cost in K segments up to data point n,

$$\tilde{C}^*_{K,n} = \min_{u_K} \underbrace{\min_{t_{K-1}} \tilde{C}^*_{K-1,t_{K-1}} + h_{t_{K-1,n}(u_K)}}_{\tilde{C}_{K,n}(u_K)}. \tag{8}$$

Note that in previous sections (equations 2,3,4,7) as well as in (8) we use a star (*) to denote the scalar optimal cost values. In the underbrace of (8) we remove the star to obtain

the function $\tilde{C}_{K,n} : \mathbb{R} \to \mathbb{R}$, which is the real-valued optimal cost as a function of the last segment mean. Minimizing the function $\tilde{C}_{K,n}(u_K)$ over all possible values of the last segment mean u_K yields the optimal cost value $\tilde{C}^*_{K,n}$.

This functional cost representation was originally proposed for the unconstrained problem (Rigaill, 2010), and it allows recursive computation of the $\tilde{C}_{K,n}$ functions via

$$\begin{aligned}
\tilde{C}_{K,n}(u_K) &= \min_{\tau\in\{K-1,\dots,n-1\}} \tilde{C}^*_{K-1,\tau} + h_{\tau,n}(u_K) && (9)\\
&= \ell(y_n, u_K) + \min\{\tilde{C}^*_{K-1,n-1}, \min_{\tau\in\{K-1,\dots,n-2\}} \tilde{C}^*_{K-1,\tau} + h_{\tau,n-1}(u_K)\} && (10)\\
&= \ell(y_n, u_K) + \min\{\tilde{C}^*_{K-1,n-1}, \tilde{C}_{K,n-1}(u_K)\}. && (11)
\end{aligned}$$

The equality (10) is obtained by removing the cost of the last data point $\ell(y_n, u_K)$ from the terms in the min. The Pruned Dynamic Programming Algorithm (PDPA) recursion (11) states that the functional cost $\tilde{C}_{K,n}(u_K)$ up to data point n can be computed using the functional cost $\tilde{C}_{K,n-1}(u_K)$ up to the previous data point $n-1$.

3.4. Functional Pruning Modified for Constraints

In this section we present our main contribution, which generalizes the functional pruning technique for the up-down constrained problem (3). We begin by defining the functional cost for the up-down constrained problem,

$$\begin{aligned}
C_{K,n}(u_K) = \min_{\substack{u_1,\dots,u_{K-1}\in\mathbb{R}\\ 0=t_0<t_1<\cdots<t_{K-1}<t_K=n}} & \sum_{k=1}^{K} h_{t_{k-1},t_k}(u_k) && (12)\\
\text{subject to} \quad & u_{k-1} \le u_k \ \forall k \in \{2,4,\dots\},\\
& u_{k-1} \ge u_k \ \forall k \in \{3,5,\dots\}.
\end{aligned}$$

The definition of the optimal cost function above removes one optimization variable, u_K, with respect to the definition of the optimal cost value $C^*_{K,n}$ (3). We have the following recursion for all $K \ge 2$:

$$\begin{aligned}
C_{K,n}(u_K) = \min_{t_{K-1},u_{K-1}} & \ C_{K-1,t_{K-1}}(u_{K-1}) + h_{t_{K-1},n}(u_K) && (13)\\
\text{subject to} \quad & \begin{cases} u_{K-1} \le u_K & \text{if } K \text{ is even,}\\ u_{K-1} \ge u_K & \text{if } K \text{ is odd.}\end{cases}
\end{aligned}$$

A key new idea of our paper is that (13) can be simplified using the min-less operator:

Definition 1 (Min-less operator) *Given any real-valued function $f : \mathbb{R} \to \mathbb{R}$, we define its min-less operator as $f^{\le}(\mu) = \min_{x\le\mu} f(x)$.*

Let us consider the case of $K = 2$. The recursion (13) simplifies to

$$C_{2,n}(u_2) = \min_{t_1} h_{t_1,n}(u_2) + \underbrace{\min_{u_1:u_1\le u_2} C_{1,t_1}(u_1)}_{C^{\le}_{1,t_1}(u_2)}. \qquad (14)$$

The min-less operator is used to write the cost as a function of a single segment mean variable u_2 (Figure 2, left). Likewise, for odd K, we need the min-more operator.

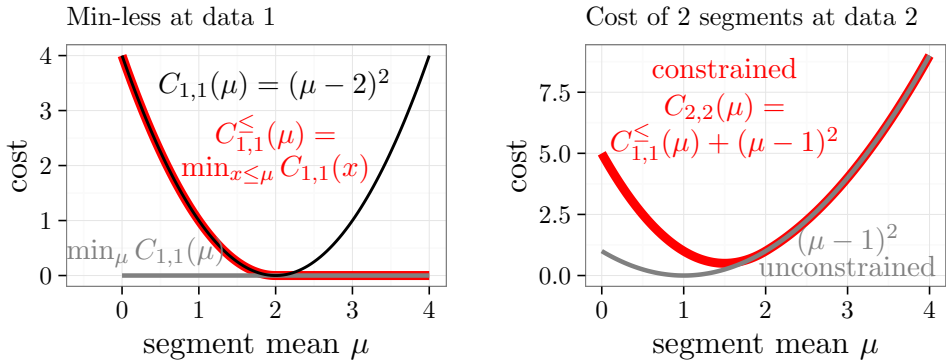


Figure 2: Comparison of previous unconstrained algorithm (grey) with new algorithm that constrains segment means to be non-decreasing (red), for the toy data set $\mathbf{y} = [2, 1, 0, 4] \in \mathbb{R}^4$ and the square loss. **Left:** rather than computing the unconstrained minimum (constant grey 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.

Definition 2 (Min-more operator) *Given any real-valued function $f : \mathbb{R} \to \mathbb{R}$, we define its min-more operator as $f^{\geq}(\mu) = \min_{x \geq \mu} f(x)$.*

The min-less and min-more operators are used in the following theorem, which states the update rules used in our proposed algorithm.

Theorem 3 (Generalized Pruned Dynamic Programming Algorithm/GPDPA) *The constrained optimal cost functions $C_{K,n}$ can be recursively computed.*

1. *For $K = 1$ we have $C_{1,1}(\mu) = \ell(y_1, \mu)$, and for the other data points $n > 1$ we have $C_{1,n}(\mu) = C_{1,n-1}(\mu) + \ell(y_n, \mu)$.*
2. *For $K > 1$ and $n = K$ we have $C_{K,K}(\mu) = \ell(y_K, \mu) + \begin{cases} C^{\leq}_{K-1,K-1}(\mu) & \text{if } K \text{ is even,} \\ C^{\geq}_{K-1,K-1}(\mu) & \text{if } K \text{ is odd.} \end{cases}$*
3. *For $K > 1$ and $n > K$ we have*

$$C_{K,n}(\mu) = \ell(y_n, \mu) + \min\{C_{K,n-1}(\mu), \begin{cases} C^{\leq}_{K-1,n-1}(\mu) & \text{if } K \text{ is even,} \\ C^{\geq}_{K-1,n-1}(\mu) & \text{if } K \text{ is odd.} \end{cases}$$

Proof Case 1 and 2 follow from the definition of the constrained functional cost (12). We prove case 3 for even K (the proof for odd K is analogous). Using the min-less operator (Definition 1), the functional cost recursion (13) simplifies to

$$C_{K,n}(u_K) = \min_{\tau \in \{K-1, \ldots, n-1\}} C^{\leq}_{K-1,\tau}(u_K) + h_{\tau,n}(u_K), \tag{15}$$

where the right term $h_{\tau,n}(u_K)$ is the cost of the last segment K, and the left term $C^{\leq}_{K-1,\tau}(u_K)$ is the cost of all previous segments. Taking out the cost of the last data point n results in

$$\begin{aligned} C_{K,n}(u_K) &= \ell(y_n, u_K) + \min \begin{cases} C^{\leq}_{K-1,K-1}(u_K) + h_{K-1,n-1}(u_K), \ldots, \\ C^{\leq}_{K-1,n-2}(u_K) + h_{n-2,n-1}(u_K), \\ C^{\leq}_{K-1,n-1}(u_K) \end{cases} \qquad (16) \\ &= \ell(y_n, u_K) + \min\{C_{K,n-1}(u_K),\, C^{\leq}_{K-1,n-1}(u_K)\}. \qquad (17) \end{aligned}$$

The final equality (17) is obtained using the definition of the constrained functional cost (12), because $C_{K,n-1}(u_K)$ is equivalent to all but the last term in the min in (16). This proves that the update rules in Theorem 3 can be used to recursively compute the constrained optimal functional cost (12). ■

3.5. Discussion of Pseudo-Code

In Algorithm 1 we provide pseudo-code that implements the update rules of Theorem 3. The first for loop initializes the optimal cost function in 1 segment, $C_{1,t}(\mu)$, for all data points $t \in \{1, \ldots, n\}$. The second for loop is over k, the number of segments. If k is even then the min-less operator is used to compute the optimal cost function $C_{k,t}(\mu)$; otherwise the min-more operator is used. The algorithm outputs the optimal cost functions $C_{k,t}(\mu)$ for all segments k and data points t.

3.6. Intuition for Speed of Functional Pruning

In this section we provide a summary of the main ideas of functional pruning for optimal changepoint detection; for a more complete discussion we refer the reader to Maidstone et al. (2017). To explain why the functional pruning approach results in a fast algorithm, we consider computing $C_{2,n}(u_2)$, the optimal cost up to data point n with second segment mean u_2. Computing this function via (14) requires taking the pointwise min of changepoint cost functions $h_{t_1,n}(u_2) + C^{\leq}_{1,t_1}(u_2)$ for all possible changepoints $t_1 \in \{1, \ldots, n-1\}$.

However, if there is a particular changepoint t_1 with a sub-optimal cost (i.e., $C_{2,n}(u_2) < h_{t_1,n}(u_2) + C^{\leq}_{1,t_1}(u_2)$ for all u_2), then that changepoint can be pruned (there is no value of the segment mean u_2 for which t_1 would be the best choice of the most recent changepoint). More precisely, let $\mathcal{T}_{2,n} = \{t_1 \mid C_{2,n}(\mu) = h_{t_1,n}(\mu) + C^{\leq}_{1,t_1}(\mu) \text{ for some } \mu\}$ be the set of changepoints which have not yet been pruned. Then the optimal cost function can also be computed by minimizing with respect to this reduced set of changepoints,

$$C_{2,n}(u_2) = \min_{t_1 \in \{1,\ldots,n-1\}} h_{t_1,n}(\mu) + C^{\leq}_{1,t_1}(\mu) = \min_{t_1 \in \mathcal{T}_{2,n}} h_{t_1,n}(\mu) + C^{\leq}_{1,t_1}(\mu). \qquad (18)$$

This pruning results in speed improvements because typically the number of candidate changepoints $|\mathcal{T}_{2,n}|$ is much smaller than n. For example in Figure 3, pruning at data point 35 results in only one candidate changepoint. In Section 5.5, we show that the empirical number of candidate changepoints in real ChIP-seq data sets is $O(\log n)$.

Algorithm 1 Generalized Pruned Dynamic Programming Algorithm (GPDPA) for up-down constrained changepoint model

Input: data $y_1, \ldots, y_n$, number of segments $K \in \mathbb{Z}_+$.
$C_{1,0}(\mu) \leftarrow 0$
for $t = 1 \ldots n$ **do**
  $C_{1,t}(\mu) \leftarrow C_{1,t-1}(\mu) + \ell(y_t, \mu)$
end for
for $k = 2, \ldots, K$ **do**
  if k is even **then**
    $C_{k,k}(\mu) \leftarrow \ell(y_k, \mu) + C^{\leq}_{k-1,k-1}(\mu)$
    for $t = k+1 \ldots, n$ **do**
      $C_{k,t}(\mu) \leftarrow \ell(y_t, \mu) + \min\{C_{k,t-1}(\mu), C^{\leq}_{k-1,t-1}(\mu)\}$
    end for
  end if
  if k is odd **then**
    $C_{k,k}(\mu) \leftarrow \ell(y_k, \mu) + C^{\geq}_{k-1,k-1}(\mu)$
    for $t = k+1 \ldots, n$ **do**
      $C_{k,t}(\mu) \leftarrow \ell(y_t, \mu) + \min\{C_{k,t-1}(\mu), C^{\geq}_{k-1,t-1}(\mu)\}$
    end for
  end if
end for
Output: For $k = 1, \ldots, K$ and $t = 1, \ldots, n$ all $C_{k,t}(\mu)$

3.7. Implementation and Computational Complexity

To implement the Generalized Pruned Dynamic Programming Algorithm (GPDPA), we use 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, typically a few intervals for each candidate changepoint. This is implemented as a linked list of FunctionPiece objects in C++ (for details see Section 8). Each element of the linked list represents a convex cost function piece, and implementation details depend on the choice of the loss function ℓ (for an example using the square loss see Section 3.8). Importantly, the min-less $C^{\leq}_{k,t}$ and min-more $C^{\geq}_{k,t}$ operators (Definitions 1 and 2) can be efficiently computed using this representation.

The functional pruning is accomplished by the min{} operation in update rule 3 of Theorem 3. For example if k is even, $C^{\leq}_{k-1,t-1}(\mu)$ is the cost if segment $k-1$ ends on data point $t-1$, and $C_{k,t-1}(\mu)$ is the cost of a changepoint before that. In the min{} operation, these two functions are compared, and pruning occurs for any cost function pieces (candidate changepoints) which are sub-optimal for all μ. For example the right panel of Figure 3 shows a data set for which all previous changepoints are pruned at $t = 35$.

We implemented the GPDPA using the Poisson loss $\ell(y, \mu) = \mu - y \log \mu$, since our application in Section 5 is on ChIP-seq non-negative count data $y \in \mathbb{Z}_+ = \{0, 1, 2, \ldots\}$. Our free/open-source C++ implementation is available as the PeakSegPDPA function in

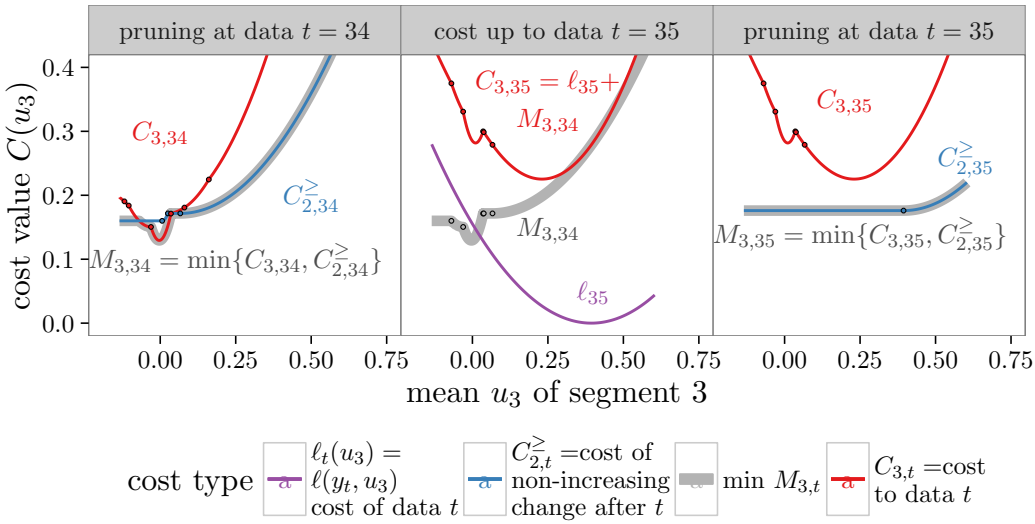


Figure 3: Demonstration of GPDPA for the up-down constrained model with 3 segments. Cost functions are stored as piecewise functions on intervals (black dots show limits between function pieces; each function piece represents a candidate changepoint). **Left:** the min $M_{3,34}$ is the minimum of two 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 the second segment ending at data point $t = 35$ is always less costly ($C^{\geq}_{2,35}$).

the PeakSegOptimal R package on CRAN.[1] Implementation details can be found in Supplementary Text 1.

The GPDPA requires computing $O(Kn)$ cost functions $C_{k,t}$. As in the original pruned dynamic programming algorithm (Rigaill, 2015), the average time complexity of the GPDPA is $O(KnI)$ where I is the average number of intervals (convex function pieces; candidate changepoints) that are used to represent a cost function. For the unconstrained problem, the theoretical maximum number of intervals is $I = O(n)$, implying a worst case time complexity of $O(Kn^2)$ (Rigaill, 2015). For the up-down constrained problem the theoretical maximum number of intervals is unknown, but in practice we have always observed $I \ll n$. For example, we investigate the empirical computational cost for the up-down constrained algorithm in Section 5.5 and observe $I = O(\log n)$, which corresponds to an overall average computational cost that is $O(Kn \log n)$.

1. `https://cran.r-project.org/package=PeakSegOptimal`

3.8. Example and Comparison with Unconstrained Algorithm

In this section we show our proposed algorithm for constrained changepoint detection differs from the previous unconstrained algorithm. We show the first few steps of the GPDPA for 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 1 segment up to data point 1:

$$C_{1,1}(u_1) = (2 - u_1)^2 = 4 - 4u_1 + u_1^2 \text{ (for } u_1 \in [0, 4]). \tag{19}$$

This function can be stored for all values of u_1 via the three real-valued coefficients (constant = 4, linear = -4, quadratic = 1). We then compute the cost of a new segment with a non-decreasing mean after the first data point (red curve on left of Figure 2),

$$C^{\leq}_{1,1}(u_2) = \begin{cases} 4 - 4u_2 + u_2^2 & \text{if } u_2 \in [0, 2],\ u_1 = u_2, \\ 0 + 0u_2 + 0u_2^2 & \text{if } u_2 \in [2, 4],\ u_1 = 2. \end{cases} \tag{20}$$

This function can be stored as a list of two intervals of mean values, each with associated real-valued coefficients. To facilitate recovery of the optimal parameters, we also store the previous segment mean u_1 and endpoint (for details see Supplementary Text 1). Note that the first interval represents the cost of an active equality constraint ($u_1 = u_2$) and the second interval represents the cost of a change up ($2 = u_1 < u_2$). In the unconstrained algorithm we would have computed the constant cost of any change (up or down) after the first data point, $\min_{u_1} C_{1,1}(u_1) = 0$ (grey curve on left of Figure 2).

By adding the cost of a non-decreasing change after the first data point $C^{\leq}_{1,1}(u_2)$ to the cost of the second data point $(u_2 - 1)^2$ we obtain the optimal cost in 2 segments up to data point 2,

$$C_{2,2}(u_2) = \begin{cases} 5 - 6u_2 + 2u_2^2 & \text{if } u_2 \in [0, 2],\ u_1 = u_2, \\ 1 - 2u_2 + 1u_2^2 & \text{if } u_2 \in [2, 4],\ u_1 = 2. \end{cases} \tag{21}$$

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 2), with an equality constraint active. This implies the optimal model up to data point 2 with 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 2), resulting in a change down from $u_1 = 2$.

3.9. Simple Data for which GPDPA is Optimal but CDPA is Not

The CDPA update rule (7) is a heuristic for solving the up-down constrained problem (3) because it is not guaranteed to compute the optimal solution. In this section we discuss two illustrative examples for which the CDPA does not compute the optimal solution, but our proposed GPDPA does.

For the set of four data points [1, 10, 14, 13] the CDPA does not compute the optimal solution for $K = 3$ segments. In fact, the CDPA returns no model when run in the forward direction on this data set, and a sub-optimal model [5.5, 5.5, 14, 13] with Poisson loss of

≈ -51.04 when run in the backward direction. In contrast, our proposed GPDPA returns the optimal model [1, 37/3, 37/3, 37/3] which has Poisson loss of ≈ -54.96. Note that the equality constraint $u_2 = u_3$ is active between the second and third segment means. This implies that the optimal model with strict inequality constraints $u_2 > u_3$ is undefined, i.e. $\forall \epsilon > 0$, $[1, 37/3+\epsilon, 37/3+\epsilon, 37/3-\epsilon]$ is not optimal because $[1, 37/3+\epsilon/2, 37/3+\epsilon/2, 37/3-\epsilon/2]$ has a lower cost.

There are also data sets for which an optimum that satisfies the strict inequality constraints exists, but the CDPA does not recover it. See Supplementary Figure 2 for a detailed discussion of the set of six data points [3, 9, 18, 15, 20, 2], for which the CDPA returns no model with $K = 5$ segments. For these data, the optimal model [6, 6, 18, 15, 20, 2] satisfies the strict inequality constraints, and is computed by our proposed GPDPA.

4. Labeled Data and Supervised Learning Algorithm

The real data analysis problem that motivates this work is the detection of peaks in ChIP-seq data (Bailey et al., 2013), which is essentially partitioning a noisy count data vector $\mathbf{y} \in \mathbb{Z}_+^n$ into peaks and background. Thus 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).

4.1. Peak Region Labels for Genomic Data Sets

More specifically, we consider the *supervised* peak detection problem, in which the peak prediction algorithm can be trained using manually determined labels that indicate presence/absence of peaks. In this context, a data set consists of M problems $\mathbf{y}_1, \ldots, \mathbf{y}_M$ along with label sets $L_1, \ldots, L_M$ that identify genomic regions with and without peaks.

Each label set $L_m = \{(\underline{p}_l, \overline{p}_l, T_l)\}_{l=1}^{|L_m|}$ is a set of tuples; each tuple (start position $= \underline{p}$, end position $= \overline{p}$, type $= T$) represents a single labeled region. Each label is a region which a biologist has determined to have presence/absence of peaks in a particular sample. Although there are only two predicted classes (positive=peaks/negative=background), there are four types of labels because the labels provide different kinds of weak/incomplete information about peak presence/absence in large regions:

noPeaks label: no peaks should be predicted anywhere in the region; $c(y_i) = 0$ for all data i in the region. False positive if any peaks are predicted in the region.

peaks label: at least one overlapping peak should be predicted somewhere in the region; $c(y_i) = 1$ for at least one i in the region. False negative if no peaks are predicted in the region.

peakStart/peakEnd labels: exactly one peak start/end should be predicted in the region. False negative if no peak start/end predicted in region, and false positive if more than one peak start/end predicted in region.

These labels can be used to compute an error rate $E[c(\mathbf{y}_m), L_m]$ and in the following we define this error rate to be the sum of false positives and false negatives. Briefly, a false

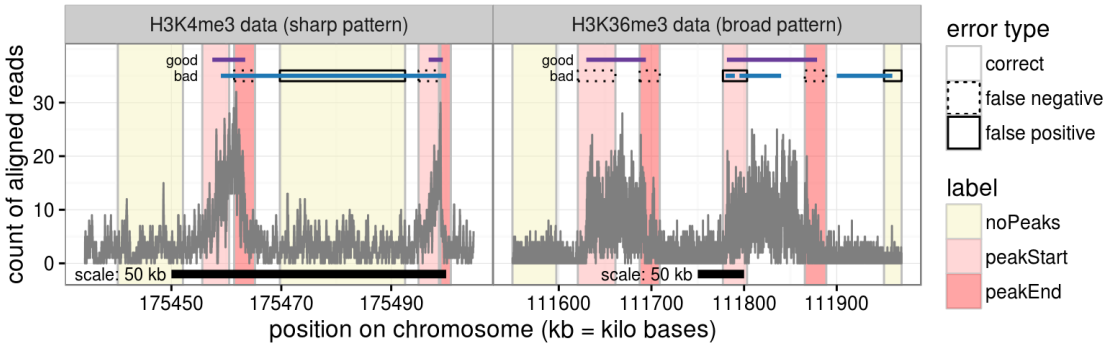


Figure 4: Labels from a biologist (colored rectangles) can be used to quantify error rates (good/bad line segments show where peaks are predicted in two different models). Error rates are computed by counting the number of incorrectly predicted labels (each noPeaks label with an overlapping peak is a false positive; each peakStart/End label requires exactly one peak start/end in that region, zero starts/ends is a false negative, and two or more starts/ends is a false positive). **Left**: a good model with two peaks results in 0 errors; a bad model with one large peak results in three errors (two false negatives and one false positive). **Right**: a good model with two peaks results in 0 errors; a bad model with three peaks results in five errors (three false negatives and two false positives).

positive is a label with too many predicted peaks, and a false negative is a label with not enough predicted peaks. Ideal peak predictions would yield zero incorrect labels (for examples see Figure 4). This error rate computation is implemented in R package PeakError; for additional details we refer the reader to (Hocking et al., 2016).

4.2. Supervised Learning Algorithm for Predicting the Penalty

For the optimal changepoint detection algorithms (CDPA, PDPA, GPDPA), predicting peaks simplifies to selecting the number of segments/changepoints. For a given problem m, let $\mathbf{y}_m \in \mathbb{Z}_+^{n_m}$ be the data, let $\hat{\mathbf{y}}_m^k \in \mathbb{R}^{n_m}$ be the mean vector with k segments computed via dynamic programming, and let $\ell(\mathbf{y}_m, \hat{\mathbf{y}}_m^k)$ be the total Poisson loss (summed over all n_m data of y_m). We use the following function to select the number of segments for a given problem m and non-negative penalty parameter λ:

$$\kappa_m(\lambda) = \arg\min_k \ell(\mathbf{y}_m, \hat{\mathbf{y}}_m^k) + \lambda \mathcal{C}(\hat{\mathbf{y}}_m^k), \tag{22}$$

where $\mathcal{C}$ is a function that measures model complexity. If λ and $\mathcal{C}$ were known in advance, then it would be computationally advantageous to reformulate the optimization problem in terms of $\lambda, \mathcal{C}$ and then solve it directly using a modification of our proposed dynamic programming algorithm (Maidstone et al., 2017; Hocking et al., 2018). However the penalty λ and model complexity $\mathcal{C}$ are generally not known in advance; there are many algorithms for choosing them and this is an active area of research (Yao, 1988; Lebarbier, 2005; Zhang and Siegmund, 2007).

In this paper we consider using the simple linear model complexity function $\mathcal{C}(\hat{\mathbf{y}}_m^k) = k$ as well as the oracle model complexity proposed by Cleynen and Lebarbier (2014),

$$\mathcal{C}(\hat{\mathbf{y}}_m^k) = k\left(1 + 4\sqrt{1.1 + \log(n_m/k)}\right)^2. \tag{23}$$

The oracle model complexity is motivated by a statistical argument that assumes a piecewise constant Poisson model, but the constants (4, 1.1) may be sub-optimal in real data that do not satisfy these assumptions.

We also investigate learned penalty functions. The main idea of penalty learning (Rigaill et al., 2013) is to compute a fixed feature vector $\mathbf{x}_m \in \mathbb{R}^d$ for each problem m, then learn a function $f(\mathbf{x}_m) \in \mathbb{R}$ that predicts problem-specific $\log \lambda$ values,

$$\underset{f}{\text{minimize}} \sum_{m=1}^{M} E\left[c(\mathbf{y}_m, \kappa_m(\exp f(\mathbf{x}_m))), L_m\right], \tag{24}$$

where $c(\mathbf{y}_m, k) \in \{0,1\}^{n_m}$ is the peak prediction vector for the model with k segments. The goal in (24) is to find the penalty function f that minimizes the total number of incorrect labels. This problem is non-convex, so we learn f using the L1-regularized linear model and convex relaxation previously described (Rigaill et al., 2013). We used $d = 365$ features which were computed in order to get a large set of features that estimate relevant properties of the data (size, quantiles, mean, variance, etc). The main idea is to compute a large set of features and then use the L1 regularization to select which ones are relevant for predicting the penalty. First we transform $\mathbf{x}$ to another vector using one of three possible operations (identity, subtract away mean, consecutive difference). Second for each resulting vector we using one of three possible element-wise transformations (identity, absolute value, square). Third, for each resulting vector we take one of four operations (sum, mean, sd, quartiles). Fourth we take one of five nonlinear element-wise transformations (identity, sqrt, log, loglog, square). The final feature vector is all combinations of the previous operations, combined with the features that result from plugging the data size n_m into each nonlinear element-wise operation of the fourth step. We have previously used this kind of feature representation/engineering for other kinds of supervised changepoint problems (Rigaill et al., 2013). We did not attempt to use any other kinds of feature engineering; this is the only set of features that we attempted to use for learning. We used the penaltyLearning R package to compute the feature matrix and learn the L1-regularized linear model.

To compare with a baseline, we also consider learning a constant function $f(\mathbf{x}_m) = \log \lambda$ for all problems m:

$$\underset{\lambda}{\text{minimize}} \sum_{m=1}^{M} E\left[c(\mathbf{y}_m, \kappa_m(\lambda)), L_m\right]. \tag{25}$$

In this case we perform the minimization using grid search over 200 λ values evenly placed on the log scale between 10^{-2} and 10^4.

5. Results on Peak Detection in ChIP-Seq Data

5.1. Benchmark Data Set

The seven labeled ChIP-seq data sets that we consider in this paper were originally described by Hocking et al. (2016), and are freely available on the web.[2] Data set names (e.g. Broad H3K36me3 AM immune) indicate experiment/pattern type (Broad H3K36me3), labeler (AM), and cell types (immune). The data consist of two different experiment types, H3K4me3 and H3K36me3. H3K4me3 is a histone modification which typically has a sharp peak pattern (10–20kb peaks). H3K36me3 is a different histone modification which typically has a broad peak pattern with longer peaks (see Figure 4). Both types of peaks are of biological interest because they indicate genomic regions with active genes (Greer and Shi, 2012). For each experiment there are several samples of different cell types (e.g. the H3K36me3 AM immune data set consists of 15 tcell samples, 5 monocyte samples, and 1 bcell sample). Accurate peak detection in these data is important in order to characterize active regions in each sample and cell type (e.g. H3K36me3 peak predicted at a particular genomic region in tcell but not in monocyte samples). Labels in these data were determined by an expert biologist, who used visual inspection of the data to determine presence or absence of significant peaks in particular genomic regions.

5.2. Algorithms to Compare and Rules for Defining Peaks

We used the following changepoint detection algorithms to compute models with $K \in \{1, \ldots, 19\}$ segments (0 to 9 peaks) on each of the 2752 labeled ChIP-seq data sets:

Generalized Pruned Dynamic Programming Algorithm (GPDPA) Proposed algorithm that computes the optimal solution to the up-down constrained problem (3). R package PeakSegOptimal.

Pruned Dynamic Programming Algorithm (PDPA) Baseline that computes the optimal solution to the unconstrained problem (2). R package Segmentor3IsBack.

Constrained Dynamic Programming Algorithm (CDPA) Baseline that computes an approximate solution to the up-down constrained problem (3). R package PeakSegDP.

Although in the up-down constrained model (3), all even-numbered segments are supposed to be peaks (with a change up before and a change down after), it is possible that an equality constraint is active (with an equal segment mean before or after). One real data example of this is shown in Figure 5, which effectively has two consecutive up changes for models with $K \in \{7, 9\}$ segments. The unconstrained problem (2) also may result in a model with several consecutive up changes (Figure 1). In general, when the model does not satisfy the strict up-down constraints, we consider three rules for defining the predicted peaks:

Join Peaks are defined by joining segments with active equality constraints. Equivalent to defining background on every segment with a change down before and a change up after; and defining peaks everywhere else.

2. `http://members.cbio.mines-paristech.fr/~thocking/chip-seq-chunk-db/`

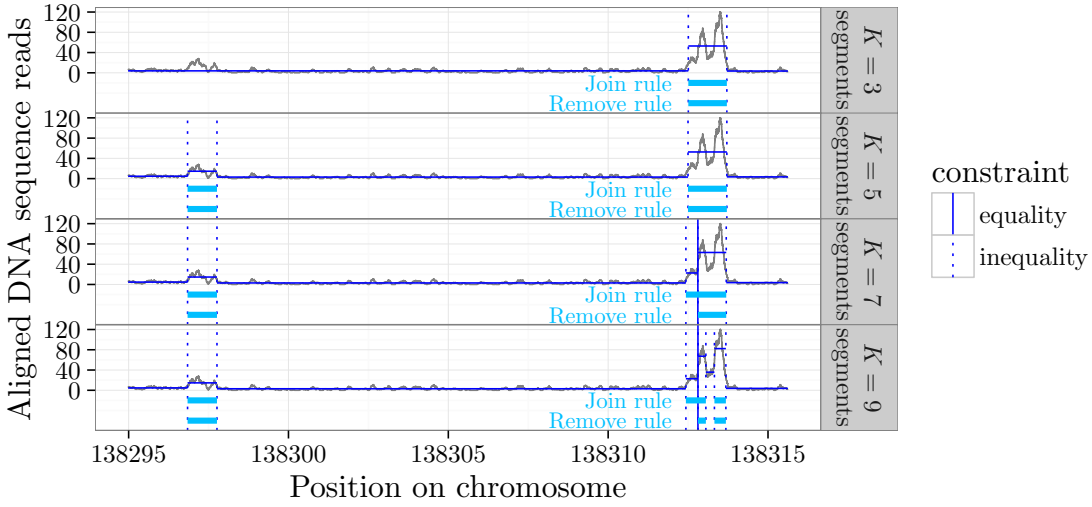


Figure 5: For piecewise constant mean models (dark blue) with active equality constraints, two different rules are used to define peaks (light blue). **Join:** segments adjacent to each equality constraint are joined to form a single peak. **Remove:** segments adjacent to each equality constraint are removed from the list of predicted peaks.

Remove A peak is removed if it occurs on a segment with an active equality constraint. Equivalent to defining a peak on every segment with a change up before and a change down after; and defining background everywhere else.

Ignore Completely ignore any model that contains at least one infeasible change (active equality constraints or consecutive changes in the same direction).

Examples of peaks defined using these rules are shown in Figure 5. Note that the Ignore rule would only consider peak models with $K \in \{3, 5\}$ segments in these data, because the other K have active equality constraints.

We additionally compare with two unsupervised baseline algorithms from the bioinformatics literature.

MACS is a heuristic algorithm with unknown time complexity from the bioinformatics literature (Zhang et al., 2008). We consider it as a baseline because it has been shown to achieve state-of-the-art peak detection accuracy for sharp H3K4me3 histone mark data (Hocking et al., 2015). We computed peaks using 53 different qvalue parameters from 0 (few peaks) to 0.8 (many peaks). We kept other parameters at default values.

HMCanBroad is a another heuristic algorithm with unknown time complexity (Ashoor et al., 2013). We consider it as a baseline because it has been shown to achieve state-of-the-art peak detection accuracy for broad H3K36me3 histone mark data (Hocking et al., 2015). We computed peaks using 112 different finalThreshold parameters from 10^{-10} (few peaks) to 10^{5} (many peaks). We used mergeDistance=1000 (recommended by authors for broad peaks), and kept other parameters at default values.

5.3. GPDPA is More Often Feasible/Optimal than Other Algorithms

In this section we compare the changepoint algorithms (PDPA, CDPA, GPDPA) in terms of optimality and feasibility for the up-down constrained segmentation problem (with strict inequality constraints). For each of the 2752 labeled count data vectors, we attempt to compute models with 0, ..., 9 peaks, so there are a total of 27520 possible models for each algorithm.

The heuristic CDPA computed the most feasible models (27469/27520=99.8%), followed by our proposed GPDPA (21278/27520=77.3%), and the unconstrained PDPA computed the fewest (8106/27520=29.4%). The heuristic CDPA was sub-optimal for 7246/27520 = 26.3% models (the proposed GPDPA was used to compute the optimal solution). For 1032/7246 of these, the optimal solution was feasible for the strict inequality constraints, and was computed by our proposed GPDPA but not the unconstrained PDPA. These results suggest that in ChIP-seq data sets, our proposed GPDPA is more accurate than the heuristic CPDA, in terms of the Poisson likelihood. Furthermore, these results suggest that GPDPA is more useful than the unconstrained PDPA, since there are many cases for which PDPA does not compute models that are feasible for the strict up-down inequality constrants (but GPDPA does).

5.4. GPDPA Fits Labels Better than Baselines in Terms of Min Train Error

In the last section we examined how the algorithms fit the data sets using the Poisson loss. However the more important measure of fit in these benchmark data is the number of incorrect labels L_m. We used the labels to compare the algorithms in terms of minimum train error, as follows. For each problem m, and a given algorithm A, we computed a sequence of P peak models $c_1^A(\mathbf{y}_m), \ldots, c_P^A(\mathbf{y}_m)$ (varying number of segments K for changepoint algorithms, qvalue/finalThreshold for baselines). For each problem m and peak model $p \in \{1, \ldots, P\}$, we computed the number of incorrect labels $E[c_p^A(\mathbf{y}_m), L_m]$. For each problem m we then compute the minimum incorrect labels over all parameters, $E_m^A = \min_p E[c_p^A(\mathbf{y}_m), L_m]$.

An algorithm A with a perfect fit to the labels would be able to achieve $E_m^A = 0$ errors for each problem m. This is not always possible in real data, due to the distribution of the labels, and the definition of the models. However, we were interested to determine which algorithms were able to achieve the fewest number of incorrect labels. In order to determine which algorithms were best able to fit the labels, we therefore compare the distribution of min error differences $E_m^a - E_m^A$ between pairs of algorithms a, A in Figure 6. Each comparison shown results in a statistically significant difference (p-value $< 10^{-4}$, two-sided paired Wilcoxon signed rank test).

Because it enforces the up-down constraints, we expected the GPDPA to be more accurate than the unconstrained PDPA. In agreement with our expectation, we observed that the up-down constrained GPDPA is indeed more accurate than the unconstrained PDPA (top panel of Figure 6), using any of the three peak definition rules (Ignore, Join, Remove). This is strong evidence that the up-down constraint is essential for accurate peak detection.

We expected the GPDPA to be just as accurate as the CDPA, and more accurate than the other baselines. In fact, we observed that the proposed GPDPA (with Remove rule) is generally more accurate than all the other algorithms (MACS, HMCanBroad, CDPA,

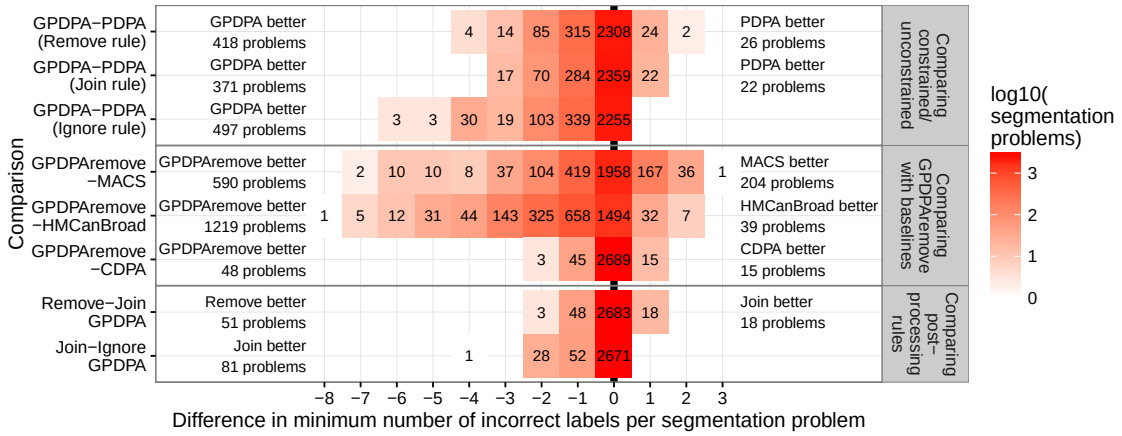


Figure 6: Distribution of differences in min label error among the 2752 labeled segmentation problems in the ChIP-seq benchmark. **Top:** up-down constrained GPDPA is more accurate than unconstrained PDPA. **Middle:** GPDPA with Remove rule is more accurate than baseline methods MACS, HMCanBroad, CDPA. **Bottom:** Remove rule is more accurate than Join, which is more accurate than Ignore.

middle panel of Figure 6). The largest difference was for HMCanBroad (GPDPA better for 1219 problems, no difference for 1494 problems, HMCanBroad better for 39 problems). The smallest difference was for CDPA (GPDPA better for 48 problems, no difference for 2689 problems, CDPA better for 15 problems). Overall these data provide strong evidence that the proposed GPDPA detects peaks more accurately than previous baseline algorithms.

When the GPDPA model has active equality constraints, we proposed three rules for defining peaks (Ignore, Remove, Join, as discussed in Section 5.2), and we did not have any expectation as to which of these rules would be most accurate in real data. We observed that the Join rule is always at least as accurate as the Ignore rule (bottom panel of Figure 6), which indicates that the Ignore rule should not be used in practice. Furthermore we observed that Remove is more accurate than Join (Remove better for 51 problems, no difference for 2683 problems, Join better for 18 problems). These data suggest that the Remove rule should be used for accurate peak detection when the GPDPA solution contains active equality constraints. So for the computational cross-validation experiments in the next sections, we used the Remove rule.

5.5. GPDPA has Log-Linear Empirical Time Complexity

Overall there are 2752 labeled count data vectors $\mathbf{y}_m$ to segment, varying in size from $n = 87$ to $n = 263169$ data. For each count data vector $\mathbf{y}_m$, we ran each algorithm (CDPA, PDPA, GDPDA) with a maximum of $K = 19$ segments (9 peaks, 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 3.7), 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

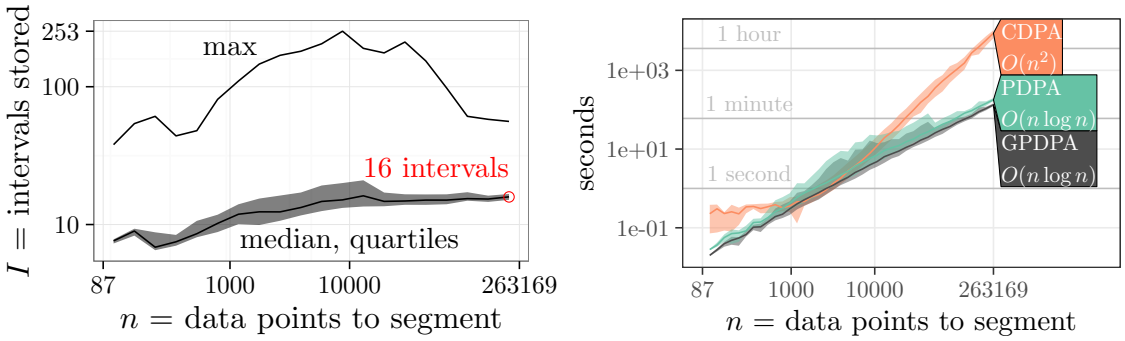


Figure 7: Empirical speed analysis on 2752 count data vectors from the histone mark ChIP-seq benchmark. For each vector we ran the GPDPA with the up-down constraint and a max of $K = 19$ segments. The expected time complexity is $O(KnI)$ where I is the average number of intervals (function pieces; candidate changepoints) stored in the $C_{k,t}$ functions. **Left**: number of intervals stored is $I = O(\log n)$ (median, inter-quartile range, and maximum over all data sets of a given size n). **Right**: GPDPA time complexity is $O(n \log n)$ (median line and min/max band).

(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 panel of Figure 7). For the largest data set ($n = 263169$), the algorithm stored only mean=16 and maximum=43 intervals (mean and maximium computed over all cost functions $C_{k,t}$ so is deterministic for a given data set). The most intervals stored in any single $C_{k,t}$ function was 253 for one data set with $n = 7776$. These results suggest that our proposed GPDPA stores on average only $O(\log n)$ 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. Since K is constant, the expected time complexity was $O(n^2)$ for the 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 panel of Figure 7). The right panel of Figure 7 also shows 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 the very large data sets that are now common in the field of genomics.

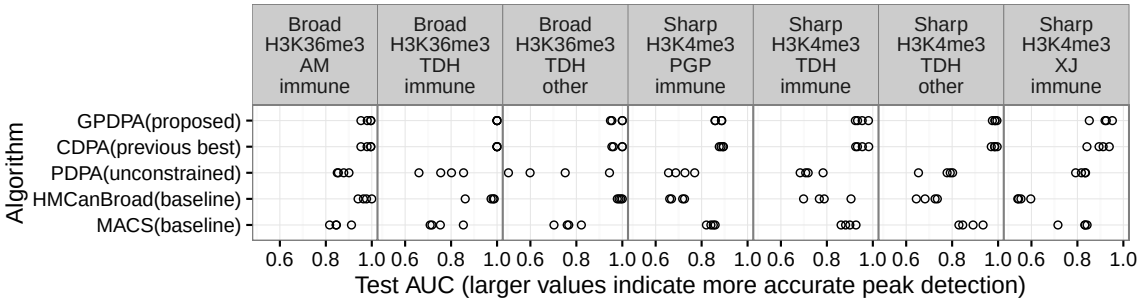


Figure 8: Four-fold cross-validation was used to estimate peak detection accuracy (black points show predicted AUC in each test fold). Each panel shows one of seven ChIP-seq data sets, labeled by pattern/experiment (Broad H3K36me3), labeler (AM), and cell types (immune). It is clear that the proposed GPDPA is just as accurate as the previous state-of-the-art CDPA, and both are more accurate than the other baseline methods.

5.6. GPDPA is More Accurate than Baselines in Terms of Test AUC

We wanted to compare the peak detection accuracy of our proposed algorithm with others from the bioinformatics literature, which typically report many false positive peaks using default parameter settings. In a typical analysis, to control the false positive rate, the default peak list is pruned by only considering the top p peaks, according to some likelihood or significance threshold. For example, the MACS algorithm of Zhang et al. (2008) uses a q-value threshold parameter, and the HMCanBroad algorithm of Ashoor et al. (2013) uses a finalThreshold parameter (higher thresholds result in more false positives). For changepoint models with a learned penalty function f (Section 4.2), the threshold is a constant $\beta \in \mathbb{R}$ which is added when selecting the number of segments $\kappa_m(\exp(f(\mathbf{x}_m)+\beta))$ (larger constants β result in larger penalties, fewer segments, and fewer false positives).

To account for this pruning step in our evaluation, we used Receiver Operating Characteristic (ROC) curve analysis. For each threshold parameter, we computed the false positive rate and true positive rate using the labels, which results in one point on the ROC curve. The area under the curve (AUC) is computed by varying the threshold parameter over its entire range (from a complete list of peaks with many false and true positives, to a completely pruned/empty list of peaks with FPR=TPR=0, see Supplementary Figure 3 for an illustration). Because the largest peak list does not necessarily predict peaks in all positive labels, we linearly extrapolate each ROC curve to TPR=FPR=1 in order to compute AUC (see Supplementary Figure 3 for an illustration of how the ROC/AUC is computed). To estimate the variance of AUC on each data set, we use four-fold cross-validation. Each labeled count data vector was randomly assigned a fold ID from 1 to 4 and then ROC curves and test AUC were computed for each fold ID.

In each of the seven data sets in the histone benchmark, we performed four-fold cross-validation and computed test AUC to estimate the accuracy of each algorithm. For the changepoint models, we learned penalty functions using the labels in each train set (Section 4.2). The previous algorithm with state-of-the-art accuracy on this benchmark was the

CDPA, the heuristic that 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 8). In five of the seven data sets, there was no significant difference in test AUC (p-value > 0.2 in two-sided paired t_3-test). In one of the two other data sets (H3K4me3 PGP immune), the GPDPA (mean AUC=0.873) was slightly less accurate than the CDPA (mean AUC=0.887, p-value=0.056); in the other data set (H3K4me3 XJ immune) the GPDPA (mean AUC=0.913) was significantly more accurate than the CDPA (mean AUC=0.897, p-value=0.02). In contrast, the unconstrained PDPA had significantly lower test AUC in all seven data sets (p-value < 0.04), because of lower true positive rates. These results provide convincing evidence that the up-down 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 optimal changepoint algorithms in all seven data sets (Figure 8). These results suggest that for detecting peaks in ChIP-seq data, the optimal changepoint algorithms are more accurate than the heuristics from the bioinformatics literature.

5.7. Learned Penalties Have Higher Test AUC than Oracle Penalties

In Section 4.2 we proposed to select the number of segments in changepoint models using two kinds of model complexity functions $\mathcal{C}$. The oracle model complexity (23) of Cleynen and Lebarbier (2014) is a relatively complex expression motivated by statistical arguments; the linear model complexity simply measures the number of segments. In this section we compare these methods in terms of test AUC.

We consider learning a one-parameter penalty function consisting of a constant penalty λ, given either the linear or oracle model complexity function. We expected the oracle model complexity to result in higher test AUC, because of its statistical motivation. The second row of Supplementary Figure 4 plots the distribution of test AUC values for one model complexity function versus the other. Contrary to our expectation, for all three changepoint algorithms (CDPA, GPDPA, PDPA), the linear and oracle penalties showed no significant difference in test AUC (mean difference from 0.00004 to 0.0006, p-value > 0.05 in paired t_{27}-test). These data indicate that, despite the statistical arguments that motivate the oracle model complexity, it is no more accurate than the simple linear model complexity for peak detection in real genomic data.

We also compare learning a penalty function with either one or multiple parameters, given the linear model complexity function. We expected higher test AUC for the penalty function with multiple parameters. The first row of Supplementary Figure 4 plots the distribution of test AUC values for one penalty function versus the other. In agreement with our expectation, for all three changepoint algorithms, the multi-parameter penalty

function has a slightly larger test AUC (mean difference from 0.003 to 0.005, p-value < 0.07). These data indicate that learning a multi-parameter penalty function should be preferred for accurate peak detection in genomic data.

5.8. Supervised is More Accurate than Unsupervised Model Training

Unsupervised algorithms are common for peak detection in genomic data, and our changepoint penalty learning method is to the best of our knowledge the first supervised method for this problem. We therefore wanted to demonstrate the superior accuracy of the supervised approach. In this section, we compare supervised and unsupervised algorithms in terms of percent correctly predicted labels in four-fold cross-validation experiments on each of the seven data sets.

First, we compared supervised single-parameter learning (grid search on a peak detection threshold) with unsupervised learning (keeping that threshold at the suggested default value). For unsupervised learning, default significance thresholds were used for HMCanBroad (finalThreshold=10) and MACS (qvalue=0.05); elbow/hinge heuristic for oracle penalty was used for PDPA/CDPA/GPDPA, as implemented in Segmentor3IsBack R package (Cleynen and Lebarbier, 2014). We expected that supervised learning would result in more accurate peak predictions. In agreement with these expectations, we observed that supervised learning had significantly higher test accuracy than unsupervised learning for almost every algorithm and data set (Supplementary Figure 5). The only exception was in data set H3K36me3 TDH other, for which the unsupervised GPDPA and CDPA were slightly more accurate (mean difference of 6.9–8.2% accuracy, not significant, p-value > 0.3 in paired t_3-test). This makes sense because that data set has the fewest labels (200) to learn from, whereas the other data sets had at least three times as many labels (630–3834). Overall these data suggest that supervised learning should be preferred for accurate peak detection, especially when there are several hundred or more labels.

We also expected that learning multi-parameter penalty functions would result in higher percent accuracy rates than single-parameter penalty functions. In agreement with this expectation, we observed that multiple parameters was at least as accurate as single parameters for every data set and algorithm (Supplementary Figure 5). For example, the GPDPA with multiple penalty parameters was significantly more accurate on two data sets (mean difference of 1–3% accuracy, p-value < 0.05 in paired t_3-test). Overall these data suggest that the current state-of-the-art for peak detection in labeled genomic data sets is achieved by the GPDPA with multi-parameter supervised penalty learning.

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). Our main contribution was showing that the functional pruning technique can be used to compute optimal changepoints subject to constraints on the directions of changes (Section 3, Theorem 3), which results in an efficient Generalized Pruned Dynamic Programming Algorithm (GPDPA).

We showed that the 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 (Section 5.5, Figure 7). We also observed that the up-down constrained GPDPA is much more accurate than the unconstrained PDPA, in terms of minimum train error (Section 5.4, Figure 6), test AUC (Section 5.6, Figure 8), and test accuracy (Section 5.8, Supplementary Figure 5). 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 proposed GPDPA, in terms of minimum train error, test AUC, and test accuracy. 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. Finally, we have recently investigated a penalized formulation of the constrained changepoint problem that results in further speedups (Hocking et al., 2018).

From a modeling perspective, the approach we implement assumes that the data starts and ends in a background segment. This is motivated by the simplicity and ease with which the results can be processed—we have a simple rule that odd numbered segments are background and even numbered ones are peaks. In our experience this does well in practice, in part because most of the genome is background noise. Our approach also assumes a Poisson model, which is simple but other loss functions may be more appropriate. The algorithmic ideas presented in this paper can be extended to include other loss functions, and data that starts or ends in a peak. In fact, inference of these models can be done with the recently released gfpop R package of Runge et al. (2020) (free/open-source implementation available on `https://github.com/vrunge/gfpop`). In the context of the supervised changepoint detection, learning the loss function (e.g. the over-dispersion parameter of the Negative Binomial) or the structure of the constraints is an interesting avenue for future work.

One can imagine jointly segmenting multiple samples, in a model that looks for common peak start/end positions across samples. Recent work in this direction has been proposed (Hocking and Bourque, 2020), but the functional pruning algorithms presented in this paper are not easily adaptable to the multi-sample case (it would require exact representation of a multi-variate cost function, which is much more complicated than the univariate cost functions presented in this paper).

The framework we have introduced for estimating changepoints when there are constraints on parameters of neighboring segments can be applied in other fields than genomics. For example, ideas from an early draft of this paper (Hocking et al., 2017) have already been used to obtain an efficient and optimal algorithm for computing a model with non-decreasing change constraints in neuro spike train data (Jewell et al., 2019). That model uses an exponential decreasing mean model for each segment (rather than the constant mean model we consider in this paper); future work may consider triangular or other shapes for the mean model in ChIP-seq data. More generally, constrained changepoint detection models will be

interesting to explore in the context of other data such as time series and statistical process monitoring.

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`

Acknowledgments

Toby Dylan Hocking and Guillaume Bourque were 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). Paul Fearnhead acknowledges EPSRC grant EP/N031938/1 (StatScale). Guillem Rigaill acknowledges an ATIGE grant from Génopole. The IPS2 benefits from the support of the LabEx Saclay Plant Sciences-SPS.

References

H Ashoor, A Hérault, A Kamoun, F Radvanyi, VB 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.

IE Auger and CE 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):e1003326, 2013.

A Barski, S Cuddapah, K Cui, T-Y Roh, DE Schones, Z Wang, G Wei, I Chepelev, and K Zhao. High-resolution profiling of histone methylations in the human genome. *Cell*, 129(4):823–837, 2007.

JD Buenrostro, B Wu, HY Chang, and WJ Greenleaf. ATAC-seq: A Method for Assaying Chromatin Accessibility Genome-Wide. *Current Protocols in Molecular Biology*, 2015.

ENCODE Project Consortium. A user's guide to the encyclopedia of DNA elements (ENCODE). *PLoS Biol*, 9(4), 2011.

C Gao, F Han, and C-H Zhang. Minimax Risk Bounds for Piecewise Constant Models. Pre-print arXiv:1705.06386, 2017.

EL Greer and Y Shi. Histone methylation: a dynamic mark in health, disease, and inheritance. *Nat Rev Genet*, 13(5):343–357, May 2012.

J Hardwick and QF Stout. Optimal reduced isotonic regression. Pre-print arXiv:1412.2844, 2014.

TD Hocking and G Bourque. Machine Learning Algorithms for Simultaneous Supervised Detection of Peaks in Multiple Samples and Cell Types. In *Proc. Pacific Symposium on Biocomputing*, volume 25, pages 367–378, 2020.

TD 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.

TD 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.

TD Hocking, G Rigaill, P Fearnhead, and G Bourque. A log-linear time algorithm for constrained changepoint detection. Pre-print arXiv:1703.03352, 2017.

TD Hocking, G Rigaill, P Fearnhead, and G Bourque. Generalized Functional Pruning Optimal Partitioning (GFPOP) for Constrained Changepoint Detection in Genomic Data. Pre-print arXiv:1810.00117, 2018.

H Hoefling. A path algorithm for the fused lasso signal approximator. *Journal of Computational and Graphical Statistics*, 19(4):984–1006, 2010. ISSN 10618600. URL `http://www.jstor.org/stable/25765384`.

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

Sean W Jewell, Toby Dylan Hocking, Paul Fearnhead, and Daniela M Witten. Fast nonconvex deconvolution of calcium imaging data. *Biostatistics*, Feb 2019. ISSN 1465-4644. doi: 10.1093/biostatistics/kxy083. URL `https://doi.org/10.1093/biostatistics/kxy083`. kxy083.

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

R Killick, P Fearnhead, and IA Eckley. Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association*, 107(500):1590–1598, 2012.

E Lebarbier. Detecting multiple change-points in the mean of gaussian process by model selection. *Signal Processing*, 85(4):717–736, 2005.

R Maidstone, TD Hocking, G Rigaill, and P Fearnhead. On optimal multiple changepoint algorithms for large data. *Statistics and Computing*, 27:519–533, 2017. 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. Pre-print 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 Rigaill, TD Hocking, J-P Vert, and F Bach. Learning sparse penalties for change-point detection using max margin interval regression. In *Proc. 30th ICML*, pages 172–180, 2013.

V Runge, TD Hocking, G Romano, F Afghah, P Fearnhead, and G Rigaill. gfpop: an R Package for Univariate Graph-Constrained Change-point Detection. Pre-print arXiv:2002.03646, 2020.

MB Rye, P Sætrom, and F Drabløs. A manually curated ChIP-seq benchmark demonstrates room for improvement in current peak-finder programs. *Nucleic Acids Research*, 39: gkq1187, 2010.

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

AM Szalkowski and CD Schmid. Rapid innovation in chip-seq peak-calling algorithms is outdistancing benchmarking efforts. *Briefings in Bioinformatics*, 12(6):626–633, 2011.

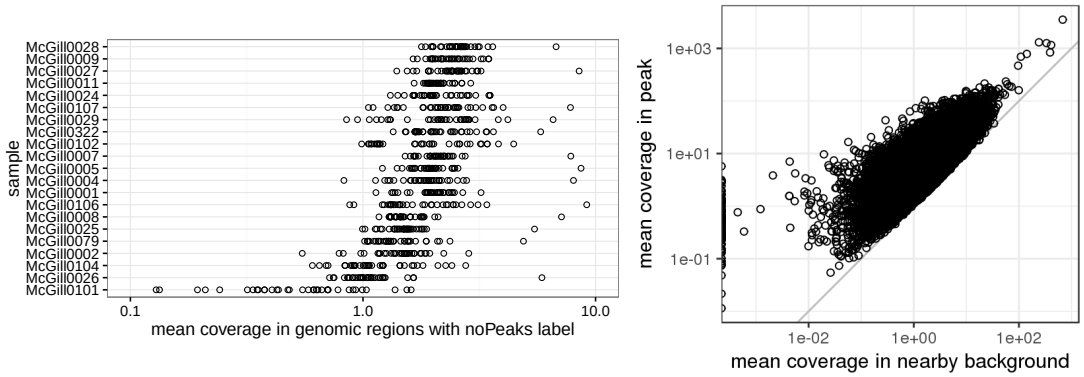
EG Wilibanks and MT Facciotti. Evaluation of Algorithm Performance in ChIP-Seq Peak Detection. *PLoS ONE*, 5(7), 2010.

Y-C Yao. Estimating the number of change-points via Schwarz' criterion. *Statistics & Probability Letters*, 6(3):181–189, February 1988.

NR Zhang and DO Siegmund. A Modified Bayes Information Criterion with Applications to the Analysis of Comparative Genomic Hybridization Data. *Biometrics*, 63:22–32, 2007.

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

Supplementary Figure 1: background level varies across a sample



Caption: Hidden Markov Models have a uniform background/noise mean level, rather than a new segment mean for every background segment (as in the model we proposed). We have observed different mean levels in different background/noise regions in the same ChIP-seq sample, which suggests that the uniform background/noise mean model would not be appropriate in these data. **Left**: we computed mean coverage in each sample and genomic region with the noPeaks label (which means a biologist has observed that there are no peaks in that region, so it must only contain background noise), and observed that the mean is highly variable between regions of the genome. For example in sample McGill0028, the mean coverage in background/noise regions ranges from 1.78 to 6.76, which suggests that a uniform/constant mean would not be a good model of these data. **Right**: for one sample, we computed peaks throughout the genome, then computed mean coverage in each peak and mean coverage in nearby background. We observed that the mean of the nearby background increases as the mean of the peak increases, which suggests that a uniform/constant mean would not be a good model of these data.

Supplementary Figure 2: details of example where CDPA fails

Consider computing the 5 segment up-down constrained model for the set of 6 data points $\mathbf{y} = [3, 9, 18, 15, 20, 2]$ using the Poisson loss $\ell(y_i, m) = m - y_i \log m$.

- The unconstrained PDPA computes the model $\mathbf{m} = [3, 9, 16.5, 16.5, 20, 2]$ which has a total Poisson loss of ≈ -109.8827. Its two increasing changes followed by two decreasing changes are not feasible for the up-down constrained problem.
- The up-down constrained GPDPA computes the model $\mathbf{m} = [6, 6, 18, 15, 20, 2]$ which has a total Poisson Loss of ≈ -108.4495. Each up change is followed by a down change, so it is feasible for the up-down constrained problem.
- The CDPA returns no feasible model with 5 segments.

To see why the CDPA fails, we give the detailed calculations of the GPDPA and CDPA below. The first cost function computed by the GPDPA is the Poisson loss of the first data point:

$$C_{1,1}(u_1) = \ell(3, u_1) = u_1 - 3 \log u_1 \tag{26}$$

The minimum of $C_{1,1}$ is at 3, so its min-less operator is convex for $u_2 \leq 3$, and constant for $u_2 \geq 3$:

$$C^{\leq}_{1,1}(u_2) = \begin{cases} C_{1,1}(u_2) = u_2 - 3 \log u_2 & \text{if } u_2 \in [2, 3],\ u_1 = u_2 \\ C_{1,1}(3) = -0.296 & \text{if } u_2 \in [3, 20],\ u_1 = 3 \end{cases} \tag{27}$$

The second cost function is the total Poisson loss of the first two data points:

$$C_{1,2}(u_1) = \ell(9, u_1) + C_{1,1}(u_1) = 2u_1 - 12 \log u_1 \tag{28}$$

The minimum of $C_{1,2}$ is at 6, so its min-less operator is convex for $u_2 \leq 6$, and constant for $u_2 \geq 6$:

$$C^{\leq}_{1,2}(u_2) = \begin{cases} C_{1,2}(u_2) = 2u_2 - 12 \log u_2 & \text{if } u_2 \in [2, 6],\ u_1 = u_2 \\ C_{1,2}(6) = -9.501 & \text{if } u_2 \in [6, 20],\ u_1 = 6 \end{cases} \tag{29}$$

The best cost in 2 segments up to data point 2 is:

$$C_{2,2}(u_2) = \ell(9, u_2) + C^{\leq}_{1,1}(u_2) = \begin{cases} 2u_2 - 12 \log u_2 & \text{if } u_2 \in [2, 3],\ u_1 = u_2 \\ u_2 - 9 \log u_2 - 0.296 & \text{if } u_2 \in [3, 20],\ u_1 = 3 \end{cases} \tag{30}$$

Note in the equation above that a non-decreasing change between data points 1 and 2 is enforced by the min-less operator $C^{\leq}_{1,1}$.

The best cost in 2 segments up to data point 3 is defined as:

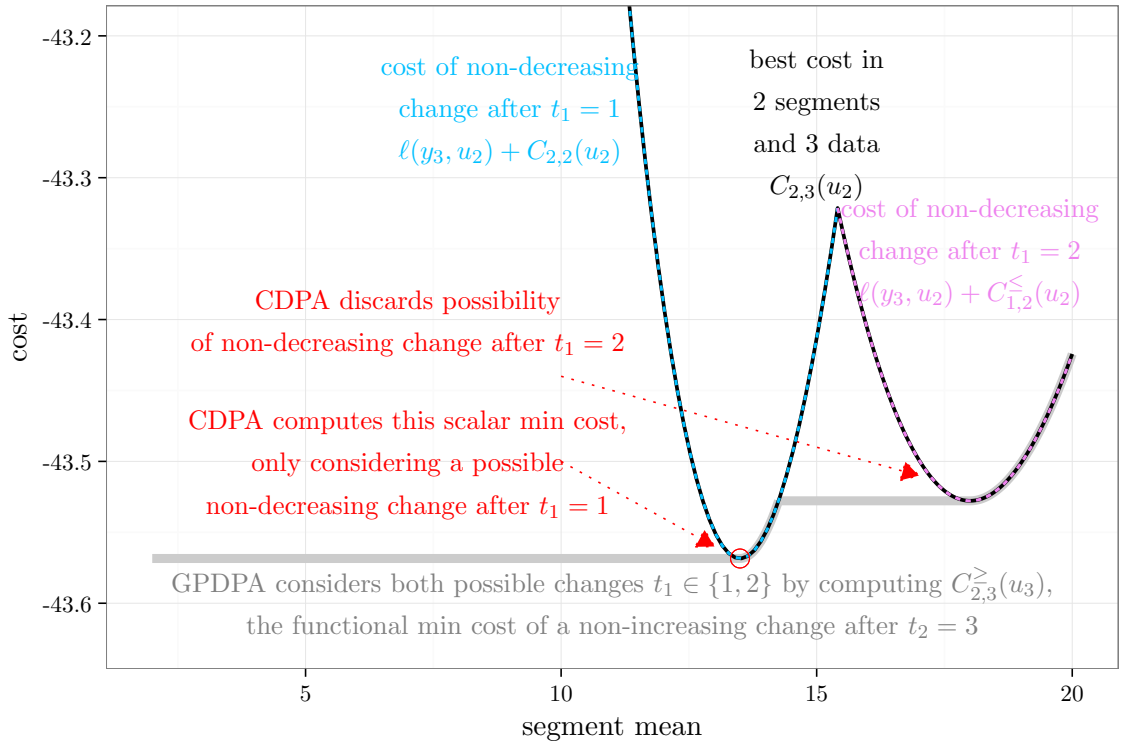
$$C_{2,3}(u_2) = \ell(18, u_2) + \min \begin{cases} C_{2,2}(u_2) & \text{if } t_1 = 1 \\ C^{\leq}_{1,2}(u_2) & \text{if } t_1 = 2 \end{cases} \tag{31}$$

The GPDPA computes the roots of $C_{2,2}(u_2) - C^{\leq}_{1,2}(u_2)$ in order to find their intersection at $u_2 \approx 15.41$, so the best cost in 2 segments up to data point 3 simplifies to:

$$C_{2,3}(u_2) = \ell(18, u_2) + \begin{cases} C_{2,2}(u_2) & \text{if } u_2 \in [2, 15.41],\ t_1 = 1 \\ C^{\leq}_{1,2}(u_2) & \text{if } u_2 \in [15.41, 20],\ t_1 = 2 \end{cases} \tag{32}$$

$$= \begin{cases} 3u_2 - 30\log u_2 & \text{if } u_2 \in [2, 3],\ u_1 = u_2,\ t_1 = 1 \\ 2u_2 - 27\log u_2 - 0.296 & \text{if } u_2 \in [3, 15.41],\ u_1 = 3,\ t_1 = 1 \\ u_2 - 18\log u_2 - 9.501 & \text{if } u_2 \in [15.41, 20],\ u_1 = 6,\ t_1 = 2 \end{cases} \tag{33}$$

Caption: The cost function above is shown as the black curve in the figure below. The part on the left ($u_2 \leq 15.41$) in blue is the cost of a non-decreasing change after the first data point ($t_1 = 1$). The part on the right in violet ($u_2 \geq 15.41$) is the cost of a non-decreasing change after the second data point ($t_2 = 2$).



The GPDPA then computes the functional min cost $C^{\geq}_{2,3}(u_3)$ for all possible values of the mean parameter u_3 (shown as grey function in plot above):

$$C^{\geq}_{2,3}(u_3) = \begin{cases} -43.57 & \text{if } u_3 \in [2, 13.5],\ u_2 = 13.5,\ t_1 = 1 \\ 2u_3 - 27\log u_3 - 0.296 & \text{if } u_3 \in [13.5, 14.25],\ u_2 = u_3,\ t_1 = 1 \\ -43.53 & \text{if } u_3 \in [14.25, 18],\ u_2 = 18,\ t_1 = 2 \\ u_3 - 18\log u_3 - 9.501 & \text{if } u_3 \in [18, 20],\ u_2 = u_3,\ t_1 = 2 \end{cases} \tag{34}$$

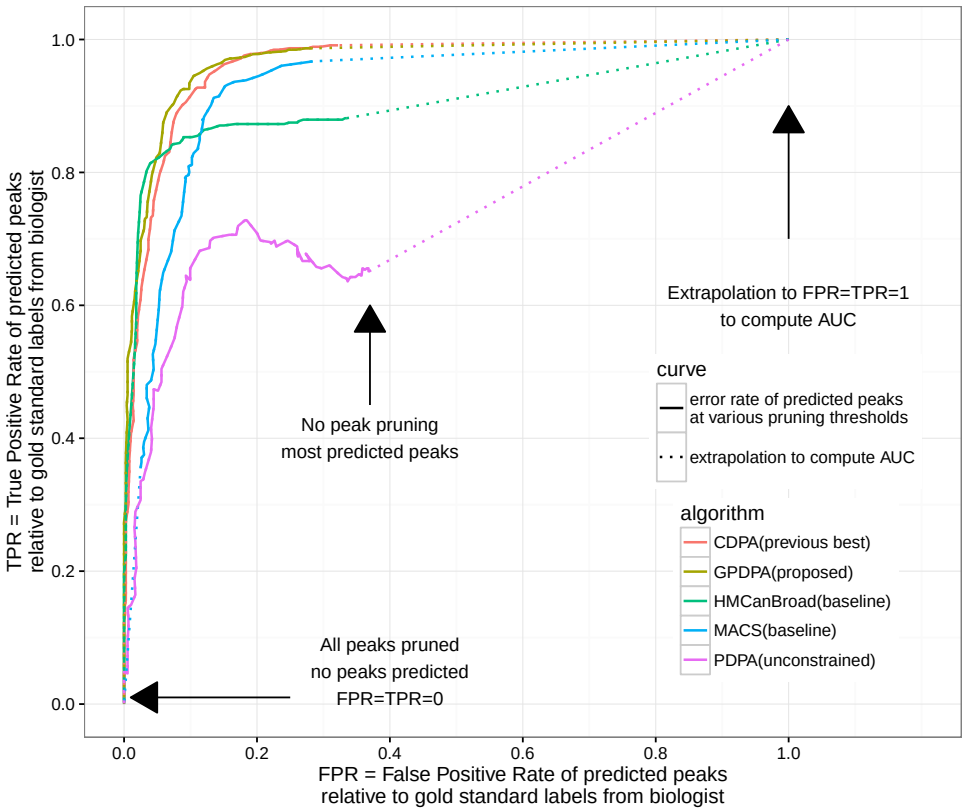
Since the optimal cost is computed for both possible changepoints, and all possible mean values, the GPDPA is able to compute the optimal solution (which occurs at $u_3 = 15, u_2 =$

$18, u_1 = 6, t_1 = 2$, but is unknown until the algorithm computes the total cost of all the data points $C_{5,6}$). In contrast, the CDPA computes a scalar min cost of a non-decreasing change after the first data point (red circle in the plot above, $u_2 = 13.5, u_1 = 3, t_1 = 1$), and discards the possibility of a non-decreasing change after the second data point (which ends up being optimal). The CDPA is thus a greedy algorithm.

Supplementary Figure 3: ROC curves for peak detection accuracy

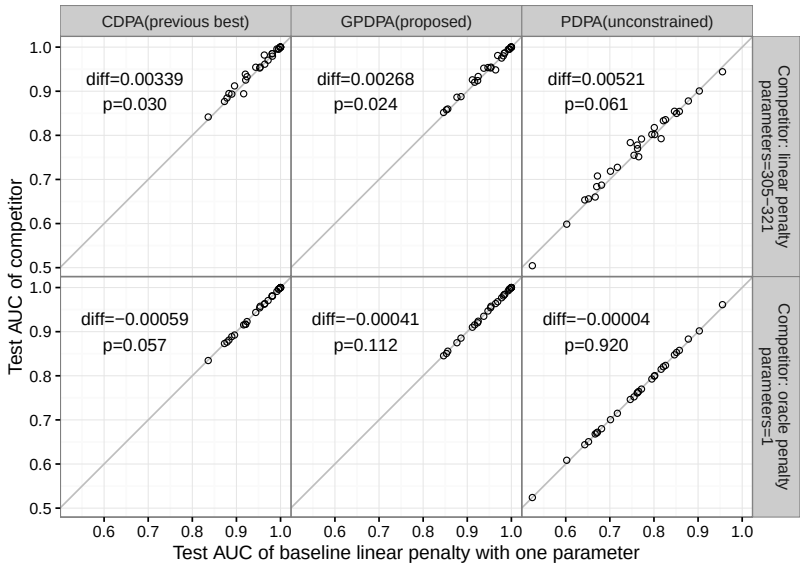
We wanted to compare the peak detection accuracy of our proposed algorithm with others from the bioinformatics literature, which typically report many false positive peaks using default parameter settings. Therefore in a typical analysis, to control the false positive rate, the default peak list is pruned by only considering the top p peaks, according to some likelihood or significance threshold. For example, the MACS algorithm uses a q-value threshold parameter, and HMCanBroad uses a finalThreshold parameter (higher thresholds result in more false positives).

Caption: To account for this pruning step in our evaluation, we used Receiver Operating Characteristic (ROC) curve analysis. For each threshold parameter, we computed the false positive rate and true positive rate using the labels, which results in one point on the ROC curve. The area under the curve (AUC) is computed by varying the threshold parameter over its entire range (from a complete list of peaks with many false and true positives, to a completely pruned/empty list of peaks with FPR=TPR=0). Note that even with the largest number of predicted peaks in each model, not all labels achieve their max possible TP and FP, because the largest peak list does not necessarily predict peaks in all labels. We therefore linearly extrapolate each ROC curve to TPR=FPR=1 in order to compute AUC (dotted lines in plot below).



Note that the ROC curves are not necessarily monotonic, because the peak pruning is not necessarily hierarchical. For example the GPDPA computes optimal models from 0 to 9 peaks for each problem; the peak present in the optimal model with 1 peak may not be present in the optimal model with 2 or more peaks.

Supplementary Figure 4: test AUC comparison between penalty functions



Caption: Test AUC of one algorithm is plotted against another algorithm, in order to visualize whether or not there are any significant differences between algorithms. Each dot represents the accuracy with respect to a single train/test split. Mean difference is shown as well as p-value in paired t-test.

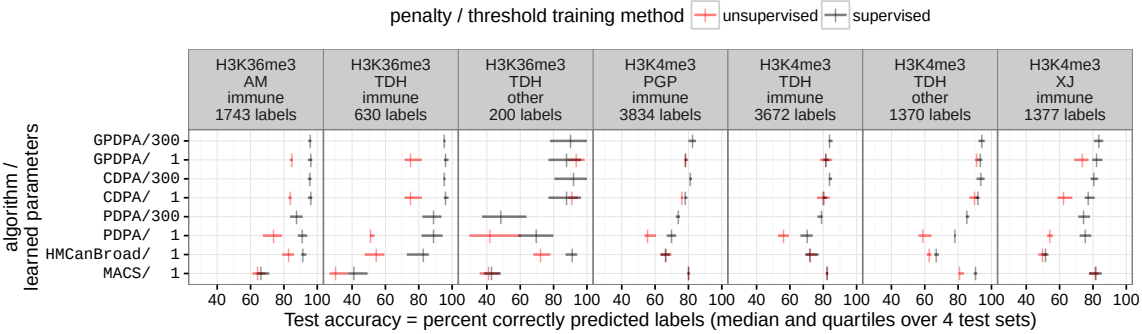
We proposed to select the number of segments in changepoint models using two kinds of model complexity functions $\mathcal{C}$. The oracle model complexity is a relatively complex expression motivated by statistical arguments; the linear model complexity simply measures the number of segments. In this section we compare these methods in terms of test AUC.

We consider learning a one-parameter penalty function consisting of a constant penalty λ, given either the linear or oracle model complexity function. We expected the oracle model complexity to result in higher test AUC, because of its statistical motivation. The second row of Supplementary Figure 4 plots the distribution of test AUC values for one model complexity function versus the other. Contrary to our expectation, for all three changepoint algorithms (CDPA, GPDPA, PDPA), the linear and oracle penalties showed no significant difference in test AUC (mean difference from 0.00004 to 0.0006, p-value > 0.05 in paired t_{27}-test). These data indicate that, despite the statistical arguments that motivate the oracle model complexity, it is not more accurate than the simple linear model complexity for peak detection in real genomic data.

We also compare learning a penalty function with either one or multiple parameters, given the linear model complexity function. We expected higher test AUC for the penalty function with multiple parameters. The first row of Supplementary Figure 4 plots the

distribution of test AUC values for one penalty function versus the other. In agreement with our expectation, for all three changepoint algorithms, the multi-parameter penalty function has a slightly larger test AUC (mean difference from 0.002 to 0.006, p-value < 0.07). These data indicate that learning a multi-parameter penalty function should be preferred for accurate peak detection in genomic data.

Supplementary Figure 5: test accuracy for peak models



Caption: Four-fold cross-validation was used to estimate peak prediction accuracy. Each panel shows one of seven ChIP-seq data sets, labeled by pattern/experiment (Broad H3K36me3), labeler (AM), cell types (immune), and number of labels. It is clear that supervised models are generally more accurate than unsupervised, and multi-parameter models are generally more accurate than single parameter models.

8. Implementation details

8.1. GPDPA pseudocode

In this section we give more detailed pseudocode for our proposed Generalized Pruned Dynamic Programming Algorithm (GPDPA). 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 pseudo-code, which describes the GPDPA for solving the up-down constrained changepoint problem with K segments.

Algorithm 2 Generalized Pruned Dynamic Programming Algorithm (GPDPA)

```
1: Input: data set y ∈ R^n, maximum number of segments K ∈ {2,...,n}.
2: Output: matrices of optimal segment means U ∈ R^{K×K} and ends T ∈ {1,...,n}^{K×K}
3: Compute min y and max ȳ of y.
4: C_{1,1} ← OnePiece(y_1, y, ȳ)
5: for data points t from 2 to n:
6:    C_{1,t} ← OnePiece(y_t, y, ȳ) + C_{1,t−1}
7: for segments k from 2 to K: for data points t from k to n: // dynamic programming
8:    MinLessOrMore ← MinLess if k is even, else MinMore
9:    min_prev ← MinLessOrMore(t − 1, C_{k−1,t−1})
10:   min_new ← min_prev if t = k, else MinOfTwo(min_prev, C_{k,t−1})
11:   C_{k,t} ← min_new + OnePiece(y_t, y, ȳ)
12: for segments k from 1 to K: // decoding for every model size k
13:   u*, t', u' ← ArgMin(C_{k,n})
14:   U_{k,k} ← u*; T_{k,k} ← t' // store mean of segment k and end of segment k − 1
15:   for segment s from k − 1 to 1: // decoding for every segment s < k
16:      if u' < ∞: u* ← u' // equality constraint active, u_s = u_{s+1}
17:      t', u' ← FindMean(u*, C_{s,t'})
18:      U_{k,s} ← u*; T_{k,s} ← t' // store mean of segment s and end of segment s − 1
```

Algorithm 2 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 7–11. Line 9 uses the MinLess (or MinMore) sub-routine to compute the temporary FunctionPieceList min_prev (which represents the function $C^{\leq}_{k-1,t-1}$ or $C^{\geq}_{k-1,t-1}$). Line 10 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 11 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 12–18. For a given model size k, the decoding begins on line 13 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 14). 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 16). The decoding of the other segments $s<k$ proceeds using the FindMean sub-routine (line 17). 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 18).

The time complexity of Algorithm 2 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 2 is $O(Kn\log n)$.

8.2. Min-less algorithm

The following sub-routines are used to implement the MinLess sub-routine.

- 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).

Pseudocode for the MinLess sub-routine is given below. The MinMore sub-routine is similar.

Algorithm 3 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 3 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.

8.3. 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 $n = 4$ 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).