Generalized Functional Pruning Optimal Partitioning (GFPOP) for Constrained Changepoint Detection in Genomic Data

Toby Dylan Hocking, Toby.Hocking@nau.edu
Guillem Rigaill, Guillem.Rigaill@inra.fr
Paul Fearnhead, p.fearnhead@lancaster.ac.uk
Guillaume Bourque, guil.bourque@mcgill.ca

October 2, 2018

Abstract

We describe a new algorithm and R package for peak detection in genomic data sets using constrained changepoint algorithms. These detect changes from background to peak regions by imposing the constraint that the mean should alternately increase then decrease. An existing algorithm for this problem exists, and gives state-of-the-art accuracy results, but it is computationally expensive when the number of changes is large. We propose the GFPOP algorithm that jointly estimates the number of peaks and their locations by minimizing a cost function which consists of a data fitting term and a penalty for each changepoint. Empirically this algorithm has a cost that is $O(N \log(N))$ for analysing data of length N. We also propose a sequential search algorithm that finds the best solution with K segments in $O(\log(K)N \log(N))$ time, which is much faster than the previous $O(KN \log(N))$ algorithm. We show that our disk-based implementation in the **PeakSegDisk** R package can be used to quickly compute constrained optimal models with many changepoints, which are needed to analyze typical genomic data sets that have tens of millions of observations.

1 Introduction

1.1 Peak detection via changepoint methods

There are many applications, particularly within genomics, that involve detecting regions that deviate from a usual/background behaviour, and where qualitatively these deviations lead to an increased mean of some measured signal. For example, ChIP-seq data measure transcription factor binding or histone modification [Barski et al., 2007]; ATAC-seq data measure open chromatin [Buenrostro et al., 2015]. In these data we have counts of aligned reads at different positions along a chromosome, and we would like to detect regions for which the count data are larger than the usual background level.

One approach to detecting these regions is through algorithms that detect changes in the mean of the data. This paper builds on recent work of Hocking et al. [2017] and presents a new changepoint algorithm, and its implementation in R. This algorithm is based on modeling count data using a Poisson distribution, and using the knowledge that we have background regions with small values and peak regions with large values. This imposes constraints on the directions of changes, with the mean of the data alternately increasing then decreasing in value. A particular challenge with genomic data is that for an algorithm to be widely used, it must scale well to large data in terms of both time and memory costs.

There are other algorithms for tackling this type of problem, for example based on hidden Markov models [Choi et al., 2009]. One drawback of such methods is that they assume the background/peak means do not change across large genomic regions, whereas such long-range changes are observed in many real data sets. For a detailed comparison of other algorithms with changepoint approaches we refer the reader to [Hocking et al., 2016]; we focus the remainder of the paper on optimal changepoint models.

1.2 Optimal changepoint models with no constraints between adjacent segment means

Denote the data by $z_1, \ldots, z_N$. We assume the data is ordered: for genomic applications the ordering will be due to position along a chromosome, for time-series data the ordering is commonly by time. The aim of changepoint analysis is to partition the data in to K segments that each contain consecutive data points, such that features of the data are common within a segment but differ between segments. The feature of the data that changes will depend on the application, but could be, for example, the mean of the data, the variance, or the distribution. Detecting changes of different features requires different statistical algorithms.

Throughout we will let K be the number of segments, with the changepoints being $0 = t_0 < t_1 < \cdots < t_{K-1} < t_K = N$. This means that the kth segment will contain data points $z_{t_{k-1}+1}, \ldots, z_{t_k}$. We denote the segment-specific parameter for the segment by m_k. For the problem of detecting changes in ChIP-seq count data, the simplest statistical model uses Poisson random variables with segment-specific mean parameters for that segment. Change detection is then an attempt to detect the points along the chromosome where the mean of the data changes.

The algorithm we present is based on detecting changes via minimizing a measure of fit to the data, with this measure of fit being the negative log-likelihood under our Poisson model. This corresponds to using the loss function $\ell(m, z) = m - z \log m$ for fitting a non-negative count data point $z \in \mathbb{Z}_+$ with a mean parameter $m \in \mathbb{R}_+$. If we know the number of segments K we can estimate the location of the segments by solving the following minimization problem,

$$\underset{\substack{\mathbf{m}\in\mathbb{R}^K \\ 0=t_0<t_1<\cdots<t_{K-1}<t_K=N}}{\text{minimize}} \quad \sum_{k=1}^{K} \sum_{i=t_{k-1}+1}^{t_k} \ell(m_k, z_i). \tag{1}$$

Optimizing by naively searching over all possible arrangements of changepoints is an expensive $O(N^K)$ time operation. However, solving (1) can be achieved efficiently using dynamic programming. The first such algorithm was the Segment Neighborhood algorithm, which computes the series of optimal segmentations with 1 to K segments in $O(KN^2)$ time [Auger and Lawrence, 1989]. The classical algorithm for solving the Segment Neighborhood problem is available in R as `changepoint::cpt.mean`. Recent research has led to faster algorithms, based on pruning the search space of the Segment Neighborhood algorithm [Rigaill, 2015, Johnson, 2013], and these algorithms empirically take $O(KN \log N)$ time. The novelty of these techniques is a functional representation of the optimal cost, which allows pruning of the $O(N)$ possible changepoints to only $O(\log N)$ candidates (while maintaining optimality). The original implementation of the PDPA was available in R as `cghseg:::segmeanCO` for the Normal homoscedastic model, but **cghseg** has been removed from CRAN as of 18 December 2017. The PDPA for the Normal homoscedastic model is now available as `jointseg::Fpsn` on Bioconductor [Pierre-Jean et al., 2015]. Cleynen and Lebarbier [2014] described a generalization of the PDPA for other likelihood/loss functions (Poisson, negative binomial, Normal heteroscedastic). These are available in R as `Segmentor3IsBack::Segmentor`.

In practice it is unusual to know how many segments there are present in the data. To estimate K it is common to use some information criteria that takes account both of the measure of fit to the data and the complexity of the segmentation model being fitted. The most natural measures of complexity are linear in the number of changepoints. Whilst it is possible to estimate K by solving the Segment Neighborhood problem for an appropriate set of changes, and calculating the value of the information criteria for each value of the number of segments, it is faster to jointly estimate both K and the changepoint locations that minimise the information criteria. The first algorithm to do so was the Optimal Partioning algorithm introduced by Jackson et al. [2005]. Optimal partitioning is an $O(N^2)$ algorithm, and can be significantly faster than Segment Neighborhood for large K.

There has also been substantial research into speeding up the optimal partitioning algorithm, using various ideas to prune the search space. In particular the Pruned Exact Linear Time (PELT) algorithm of Killick et al. [2012], which is implemented within `changepoint::cpt.mean`, and Functional Pruning Optimal Partitioning (FPOP) of Maidstone et al. [2016] which is available in R as `fpop::Fpop` [Rigaill and

Problem	No changepoint pruning	Functional pruning
Segment Neighborhood K segments	Dynamic Prog. Algo. (DPA) Optimal, $O(KN^2)$ time Auger and Lawrence [1989] **changepoint**	Pruned DPA (PDPA) Optimal, $O(KN \log N)$ time Rigaill [2015] **jointseg**
Optimal Partitioning Penalty λ	Optimal Partitioning Algorithm Optimal, $O(N^2)$ time Jackson et al. [2005]	FPOP Optimal, $O(N \log N)$ time Maidstone et al. [2016] **fpop**

Table 1: Previous work on algorithms for optimal changepoint detection with no constraints between adjacent segment means.

Hocking, 2016]. These algorithms have a computational cost of $O(N)$ if K increases linearly with N. The FPOP algorithm has a computational cost that is empirically $O(N \log N)$ in situations where K increases sub-linearly with N. See Table 1 for a summary of the different dynamic programming algorithms and implementations.

Whilst solving the optimal partitioning problem is faster than solving (1) for a range of K, the drawback is that you only get a single segmentation for a single value K of the number of segments. Furthermore the choice of penalty that you impose with the information criteria – which corresponds the improvement in fit to the data needed to add an additional changepoint – can be hard to tune and have an important impact on the accuracy of the estimate of the number of changepoints. One way to ameliorate this concern is to find segmentations for a range of penalties, which can be done efficiently [Haynes et al., 2017].

There are alternative approaches to fitting changepoint models, the most common of which are based on specifying a test for a single change and then repeatedly applying this test to identify multiple changepoints. Such approaches can be applied more widely than the dynamic programming based approaches described above, and often have strong computational performance with algorithms that are $O(N \log N)$ for the Segment Neighborhood problem. In situations where both procedures can be used, these methods are often identical if we wish to identify at most one changepoint. The advantage that the dynamic programming approaches have is that they jointly detect multiple changepoints which can lead to more accurate estimates [see e.g. Maidstone et al., 2016]. Several of these alternative algorithms are available in R. For example, the **wbs** package implements the wild binary segmentation method of Fryzlewicz [2014]. An efficient implementation of the classical binary segmentation heuristic is available as `fpop::multiBinSeg`. The **stepR** package implements the SMUCE algorithm for multiscale changepoint inference [Frick et al., 2014].

1.3 Models with inequality constraints between adjacent segment means

The models discussed above are unconstrained in the sense that there are no constraints between mean parameters m_k on different segments. However, as described above, constraints can be useful when data need to be interpreted in terms of pre-defined domain-specific states. In the ChIP-seq application the changepoint model needs to be interpreted in terms of peaks (large values which represent protein binding/modification) and background (small values which represent noise).

In this context, Hocking et al. [2015] introduced a $O(KN^2)$ Constrained Dynamic Programming Algorithm (CDPA) for fitting a model where up changes are followed by down changes, and vice versa (Table 2). These constraints ensure that odd-numbered segments can be interpreted as background, and even-numbered segments can be interpreted as peaks. Although the CDPA provides a sub-optimal solution to the Segment Neighborhood problem in $O(KN^2)$ time, Hocking et al. [2016] showed that it achieves state-of-the-art peak detection accuracy in a benchmark of ChIP-seq data sets.

Because the quadratic time complexity of the CDPA limits its application to relatively small data sets, Hocking et al. [2017] proposed to generalize the functional pruning method for changepoint models with constraints between adjacent segment means. The resulting Generalized Pruned Dynamic Programming

	No changepoint pruning	Functional pruning
Segment Neighborhood K segments	Constrained DPA Sub-optimal, $O(KN^2)$ Hocking et al. [2015] **PeakSegDP**	Generalized PDPA Optimal, $O(KN \log N)$ Hocking et al. [2017] **PeakSegOptimal**
Optimal Partitioning Penalty λ		Generalized FPOP Optimal, $O(N \log N)$ **This work** **PeakSegDisk**

Table 2: Algorithms for optimal changepoint detection with up-down constraints on adjacent segment means. Previous work is limited to solvers for the Segment Neighborhood problem; this paper presents Generalized Functional Pruning Optimal Partitioning (GFPOP), Algorithm 1.

Algorithm (GPDPA) reduces the number of candidate changepoints from $O(N)$ to $O(\log N)$ while enforcing the constraints and maintaining optimality. The GPDPA computes the optimal solution to the up-down constrained Segment Neighborhood problem in $O(KN \log N)$ time. The **PeakSegOptimal** R package provides an in-memory solver for the up-down constrained Segment Neighborhood model [Hocking et al., 2017].

1.4 Contributions

This paper presents two new algorithms for constrained optimal changepoint detection (Section 3), along with an analysis of their empirical time/space complexity in a benchmark of genomic data (Section 4). The algorithms are implemented in the R package **PeakSegDisk** on GitHub.[1]

First, we present a new algorithm for solving the Optimal Partitioning problem with up-down constraints between adjacent segment means (GFPOP, Algorithm 1). The fastest existing algorithm for the up-down constrained changepoint model was the $O(KN \log N)$ solver for the Segment Neighborhood problem (Table 2). In large genomic data sets, we are only interested in models with many segments/changepoints, so it is a waste of time and space to compute all models from 1 to K segments using Segment Neighborhood algorithms. Our proposed GFPOP algorithm solves the Optimal Partitioning problem, so yields one optimal model with K segments (without having to compute the models from 1 to $K-1$ segments). We show that the empirical complexity of our GFPOP implementation is $O(N \log N)$ time, $O(N \log N)$ space, and $O(\log N)$ memory, which makes it possible to compute optimal models with many peaks for typical genomic data sets on common laptop computers.

Although solving the Optimal Partitioning problem is faster by a factor of $O(K)$, the user is unable to directly choose the number of segments K. The user inputs a penalty λ, and gets one of the optimal changepoint models as output. Thus, we also propose a sequential search (Algorithm 2) which computes the optimal model for a specified number of segments K. It repeatedly calls GFPOP to solve Optimal Partitioning with different penalties λ, until it finds the maximum likelihood model with at most K segments. We empirically show that the sequential search only requires $O(\log K)$ evaluations of GFPOP. Overall the proposed algorithm is thus $O(N \log(N) \log(K))$ time, $O(N \log N)$ disk, $O(\log N)$ memory. In an analysis of benchmark genomic data sets, we show that this algorithm can compute an optimal model with $O(\sqrt{N}) > 1000$ peaks for $N = 10^7$ data using only hours of compute time and gigabytes of storage (which is much less than weeks/terabytes which would be required for the Segment Neighborhood solver).

[1] `https://github.com/tdhock/PeakSegDisk`

2 Statistical models and optimization problems

2.1 Unconstrained Optimal Partitioning problem

Define our loss function to be the Poisson loss, $\ell(m, z) = m - z \log m$, and let $\lambda > 0$ be a penalty for adding a changepoint. Then we can infer the number of segments and the location of the changes by solving the Optimal Partitioning problem

$$\underset{\mathbf{m}\in\mathbb{R}^N}{\text{minimize}} \quad \sum_{i=1}^{N} \ell(m_i, z_i) + \lambda \sum_{i=1}^{N-1} I(m_i \neq m_{i+1}). \tag{2}$$

The first term measures fit to the data, and the second term measures model complexity, which is proportional to the number of changepoints. The non-negative penalty $\lambda \in \mathbb{R}_+$ controls the tradeoff between the two objectives (it is a tuning parameter that must be fixed before solving the problem). Larger penalty λ values result in models with fewer changepoints/segments. The extreme penalty values are $\lambda = 0$ which yields N segments ($N-1$ changepoints), and $\lambda = \infty$ which yields 1 segment (0 changepoints).

Below we write an equivalent version of the Optimal Partitioning problem, in terms of changepoint variables c_i and state variables s_i:

$$\underset{\substack{\mathbf{m}\in\mathbb{R}^N, \mathbf{s}\in\{0\}^N \\ \mathbf{c}\in\{0,1\}^{N-1}}}{\text{minimize}} \quad \sum_{i=1}^{N} \ell(m_i, z_i) + \lambda \sum_{i=1}^{N-1} I(c_i = 1) \tag{3}$$

$$\begin{aligned} \text{subject to} \quad & \text{no change: } c_i = 0 \Rightarrow m_i = m_{i+1} \text{ and } s_i = s_{i+1}, \\ & \text{change: } c_i = 1 \Rightarrow m_i \neq m_{i+1} \text{ and } (s_i, s_{i+1}) = (0, 0). \end{aligned} \tag{4}$$

Note that the state s_i and changepoint c_i variables could be eliminated from the optimization problem — $s_i = 0$ and $c_i = I(m_i \neq m_{i+1})$ for all i. We include them in problem (3) in order to show the relationship with the problem in the next section, with constraints between adjacent segment means.

Hocking et al. [2017] proposed to use a graph to represent a constrained changepoint model. The graph that corresponds to problem (3) is shown in Figure 1, left. In such graphs, nodes represent possible values of state variables s_i and edges represent possible changepoints $c_i \neq 0$. Each edge/changepoint corresponds to a constraint such as (4).

2.2 Optimal Partitioning problem with up-down constraints between adjacent segment means

For genomic data such as ChIP-seq [Barski et al., 2007], it is desirable to have a changepoint model which is interpretable in terms of peaks (large values) and background noise (small values). We therefore propose a model based on the graph shown in Figure 1, right. It has two nodes/states: $s = 0$ for background, and $s = 1$ for peaks. It has two edges/changes: $c = 1$ for a non-decreasing change from background $s = 0$ to a peak $s = 1$, and $c = -1$ for a non-increasing change from a peak $s = 1$ to background $s = 0$. Furthermore, the model is constrained to start and end in the background state (because peaks are not present at the boundaries of genomic data sequences). Maximum likelihood inference in this model corresponds to the following minimization problem:

$$F(\lambda) = \min_{\substack{\mathbf{m}\in\mathbb{R}^N, \ \mathbf{s}\in\{0,1\}^N \\ \mathbf{c}\in\{-1,0,1\}^{N-1}}} \quad \sum_{i=1}^{N} \ell(m_i, z_i) + \lambda \sum_{i=1}^{N-1} I(c_i = 1) \tag{5}$$

$$\begin{aligned} \text{subject to} \quad & \text{no change: } c_i = 0 \Rightarrow m_i = m_{i+1} \text{ and } s_i = s_{i+1}, \\ & \text{non-decreasing change: } c_i = 1 \Rightarrow m_i \leq m_{i+1} \text{ and } (s_i, s_{i+1}) = (0, 1), \\ & \text{non-increasing change: } c_i = -1 \Rightarrow m_i \geq m_{i+1} \text{ and } (s_i, s_{i+1}) = (1, 0), \\ & \text{start and end down: } s_1 = s_N = 0. \end{aligned}$$

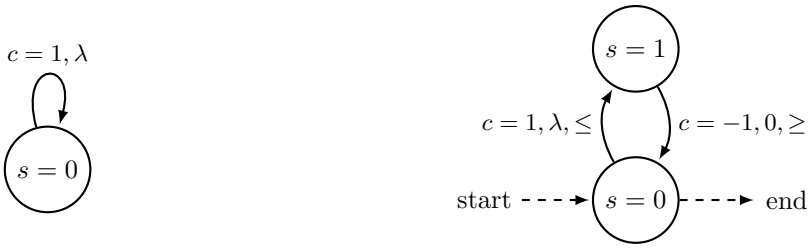


Figure 1: State graphs for two changepoint models. Nodes represent states and solid edges represent changepoints. **Left:** one-state model with no constraints between adjacent segment means, problem (3). **Right**: two-state model with up-down constraints between adjacent segment means, problem (5). State $s = 0$ represents background noise (small values) whereas state $s = 1$ represents peaks (large values). Constraint $c = 1$ enforces a non-decreasing change via the min-less operator ($\leq$) with a penalty of λ; $c = -1$ enforces a non-increasing change via the min-more operator ($\geq$) with a penalty of 0. The model is additionally constrained to start and end in the background noise $s = 0$ state ($s_1 = s_N = 0$).

Note how the problem (5) with up-down constraints is of the same form as the previous unconstrained problem (3). Again there is one constraint for every edge/changepoint in the state graph (Figure 1). The difference is that in problem (5), we have inequality constraints between adjacent segment means (e.g. when $c_i = 1$, we must have a non-decreasing change in the mean $m_i \leq m_{i+1}$). Another difference is the model complexity in problem (5) is the total number of $c_i = 1$ non-decreasing changes, which is equivalent to the number of peak segments P, and is linear in the total number of segments $K = 2P + 1$ and changes $K - 1 = 2P$.

The solution to the Optimal Partitioning problem (5) can be computed by first solving the Segment Neighborhood version of the problem [Maidstone et al., 2016]. In R the **PeakSegDP** package provides a sub-optimal solution in $O(KN^2)$ time, and the **PeakSegOptimal** package provides an optimal solution in $O(KN \log N)$ time. However in genomic data the number of peaks/segments K increases with N, so it is intractable to solve the Segment Neighborhood problem because both N and K are large. Therefore in the next section we propose a new algorithm for directly solving the constrained Optimal Partitioning problem (5), which can yield a large number of peaks in $O(N \log N)$ time.

3 Algorithms and Software

3.1 Generalized Functional Pruning Optimal Partitioning (GFPOP)

In this section we propose a generalization of the FPOP algorithm [Maidstone et al., 2016] which allows optimal inference in models with inequality constraints between adjacent means, such as problem (5). In particular we implemented the optimal changepoint model using the Poisson loss and the up-down constraints. The state graph (Figure 1, right) can be converted into a directed acyclic graph (Figure 2) that represents the dynamic programming updates required to solve problem (5). Each node in the computation graph represents an optimal cost function, and each edge represents an input to the min{} operation in the dynamic programming equations (12) and (13) below.

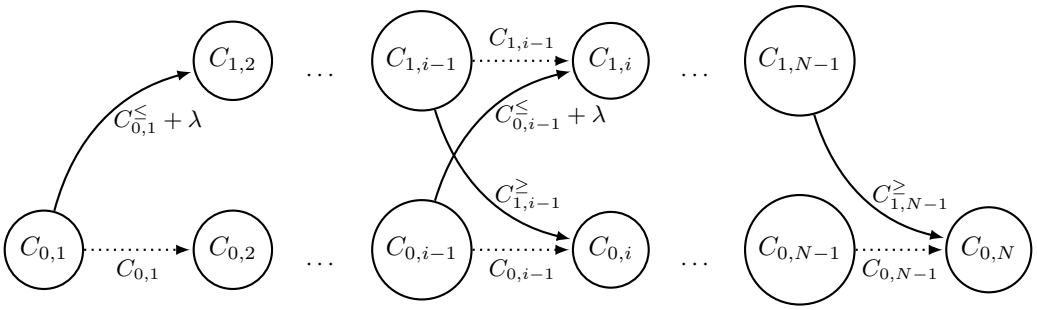


Figure 2: Directed acyclic graph (DAG) representing dynamic programming computations (Algorithm 1) for changepoint model with up-down constraints between adjacent segment means. Nodes in the graph repesent cost functions, and edges represent inputs to the the MinOfTwo sub-routine (solid=changepoint, dotted=no change). There is one column for each data point and one row for each state: the optimal cost of the peak state $s = 1$ at data point i is $C_{1,i}$ (top row); the optimal cost of the background noise state $s = 0$ is $C_{0,i}$ (bottom row). There is only one edge going to $C_{0,2}$ and $C_{1,2}$ because the model is constrained to start in the background noise state ($s_1 = 0$).

More precisely, we define the optimal cost of mean μ in state σ at any data point $\tau \in \{1, \ldots, N\}$ to be

$$C_{\sigma,\tau}(\mu) = \min_{\substack{\mathbf{m}\in\mathbb{R}^\tau,\ \mathbf{s}\in\{0,1\}^\tau \\ \mathbf{c}\in\{-1,0,1\}^{\tau-1}}} \sum_{i=1}^{\tau} \ell(m_i, z_i) + \lambda \sum_{i=1}^{\tau-1} I(c_i = 1) \tag{6}$$

$$\begin{aligned} \text{subject to} \quad & c_i = 0 \Rightarrow m_i = m_{i+1} \text{ and } s_i = s_{i+1}, \\ & c_i = 1 \Rightarrow m_i \leq m_{i+1} \text{ and } (s_i, s_{i+1}) = (0, 1), \\ & c_i = -1 \Rightarrow m_i \geq m_{i+1} \text{ and } (s_i, s_{i+1}) = (1, 0), \\ & s_1 = s_N = 0, \\ & m_\tau = \mu,\ s_\tau = \sigma. \end{aligned} \tag{7}$$

Note how the objective and constraints above are identical to the up-down constrained Optimal Partitioning problem (5) up to $\tau - 1$ data points, but with two added constraints at data point τ (7). At data point τ the mean is constrained to be $m_\tau = \mu$ and the state is constrained to be $s_\tau = \sigma$. The optimal cost $C_{\sigma,\tau}(\mu)$ is a real-valued function that must be computed by minimizing over all previous means $m_1, \ldots, m_{\tau-1}$, states $s_1, \ldots, s_{\tau-1}$, and changes $c_1, \ldots, c_{\tau-1}$. It can be computed recursively using the dynamic programming updates that we propose below.

The algorithm begins by initializing the optimal cost of the background state at the first data point,

$$C_{0,1}(\mu) = \ell(\mu, z_1). \tag{8}$$

The computations for the second data point are also special, because the model is constrained to start in the background state $s_1 = 0$. To get to the background state $s_2 = 0$ at the second data point requires no change ($c_1 = 0$), with a cost of

$$C_{0,2}(\mu) = C_{0,1}(\mu) + \ell(\mu, z_2). \tag{9}$$

Similarly, to get to the peak state $s_2 = 1$ at the second data point requires a non-decreasing change ($c_1 = 1$), with a cost of

$$C_{1,2}(\mu) = \min_{m_1 \leq \mu} C_{0,1}(m_1) + \lambda + \ell(\mu, z_2) = C^{\leq}_{0,1}(\mu) + \lambda + \ell(\mu, z_2). \tag{10}$$

Note that we were able to re-write the optimal cost function in terms of a single variable μ by using the min-less operator,

$$f^{\leq}(\mu) = \min_{x \leq \mu} f(x). \tag{11}$$

The min-less operator was introduced by Hocking et al. [2017] in order to compute the optimal cost in the functional pruning algorithm that solves the Segment Neighborhood version of this problem.

More generally, the dynamic programming update rules can be derived from the computation graph (Figure 2). The optimal cost of the peak state $s = 1$ at data $i > 2$ is

$$C_{1,i}(\mu) = \ell(\mu, z_i) + \min\{C_{1,i-1}(\mu),\, C^{\leq}_{0,i-1}(\mu) + \lambda\}. \tag{12}$$

Note how the inputs to the min{} operation are the same as the edges leading to the $C_{1,i}$ node in the computation graph (Figure 2).

Similarly, the optimal cost of the background state $s = 0$ is

$$C_{0,i}(\mu) = \ell(\mu, z_i) + \min\{C_{0,i-1}(\mu),\, C^{\geq}_{1,i-1}(\mu) + \lambda\}, \tag{13}$$

where the min-more operator is defined as

$$f^{\geq}(\mu) = \min_{x \geq \mu} f(x). \tag{14}$$

These dynamic programming computations are summarized in Algorithm 1, Generalized Functional Pruning Optimal Partitioning. The key to implementing the algorithm is to use a PiecewiseFunction data structure that can exactly represent an optimal cost function $C_{s,i}$. In the case of the Poisson loss, each $C_{s,i}(\mu)$ is a piecewise function where each piece is of the form $\alpha\mu + \beta \log \mu + \gamma$. Therefore the optimal cost can be stored as a list of intervals of $\mu \in [\text{MIN}, \text{MAX}]$, each with coefficients α, β, γ.

Algorithm 1 Generalized Functional Pruning Optimal Partitioning (GFPOP) for changepoint model with up-down constraints between adjacent segment means.

1: Input: data set $\mathbf{z} \in \mathbb{R}^N$, penalty constant $\lambda \geq 0$.
2: Output: vectors of optimal segment means $U \in \mathbb{R}^N$ and ends $T \in \{1, \ldots, N\}^N$
3: Initialize $2 \times N$ empty PiecewiseFunction objects $C_{s,i}$ either in memory or on disk.
4: Compute min $\underline{z}$ and max $\overline{z}$ of $\mathbf{z}$.
5: $C_{0,1} \leftarrow \text{OnePiece}(z_1, \underline{z}, \overline{z})$
6: for data point i from 2 to N: // dynamic programming
7: $\quad M_1 \leftarrow \lambda + \text{MinLess}(i-1, C_{0,i-1})$ //cost of non-decreasing change
8: $\quad C_{1,i} \leftarrow \text{MinOfTwo}(M_1, C_{1,i-1}) + \text{OnePiece}(z_i, \underline{z}, \overline{z})$
9: $\quad M_0 \leftarrow \text{MinMore}(i-1, C_{1,i-1})$ //cost of non-increasing change
10: $\quad C_{0,i} \leftarrow \text{MinOfTwo}(M_0, C_{0,i-1}) + \text{OnePiece}(z_i, \underline{z}, \overline{z})$
11: $\text{mean}, \text{prevEnd}, \text{prevMean} \leftarrow \text{ArgMin}(C_{0,n})$ // begin decoding
12: $\text{seg} \leftarrow 1$; $U_{\text{seg}} \leftarrow \text{mean}$; $T_{\text{seg}} \leftarrow \text{prevEnd}$
13: while prevEnd > 0:
14: $\quad$ if prevMean $< \infty$: mean $\leftarrow$ prevMean
15: $\quad$ if seg is odd: cost $\leftarrow C_{1,\text{prevEnd}}$ else $C_{0,\text{prevEnd}}$
16: $\quad \text{prevEnd}, \text{prevMean} \leftarrow \text{FindMean}(\text{mean}, \text{cost})$
17: $\quad \text{seg} \leftarrow \text{seg} + 1$; $U_{\text{seg}} \leftarrow \text{mean}$; $T_{\text{seg}} \leftarrow \text{prevEnd}$

Discussion of pseudocode. Algorithm 1 begins on line 3 by initializing the array $C_{s,i}$ of optimal cost functions (either in memory or on disk). It then computes the min $\underline{z}$ and max $\overline{z}$ of the data (line 4) and uses the OnePiece sub-routine to initialize the optimal cost at the first data point (line 5). Since the Poisson loss is $\ell(\mu, z_1) = \mu - z_1 \log \mu$, this first optimal cost function is represented as the single function piece with interval/coefficients $(\alpha = 1, \beta = -z_1, \gamma = 0, \text{MIN} = \underline{z}, \text{MAX} = \overline{z})$.

The dynamic programming recursion in this algorithm is a loop over data points i (line 6). To compute $C_{1,i}$, the penalty constant λ is added to all of the result of MinLess (line 7), before computing MinOfTwo and adding the cost of the new data point (line 8). The computation for $C_{0,i}$ is similar, but uses MinMore

and does not add the penalty λ (lines 9–10). The details about how the MinLess/MinMore/MinOfTwo sub-routines process the PiecewiseFunction objects have been described previously [Hocking et al., 2017].

After computing the optimal cost functions, the decoding of optimal parameters occurs on lines 11–17. The last segment mean and second to last segment end are first stored on line 12 in (U_1, T_1). For each other segment i, the mean and previous segment end are stored on line 17 in (U_i, T_i). Note that there should be space to store (U_i, T_i) parameters for up to N segments. In practice our implementation writes these parameters to a text output file on disk.

Computational complexity. The complexity of Algorithm 1 is $O(NI)$, where I is the mean number of intervals (function pieces) that are used to represent the $C_{s,i}$ cost functions. Theoretically there are some pathological data sets for which the algorithm computes $I = O(N)$ intervals, which results in the worst-case complexity of $O(N^2)$. Since the number of intervals in real data is empirically $I = O(\log N)$ (see Figure 5), the overall complexity of Algorithm 1 is on average $O(N \log N)$. Using disk-based storage its complexity is $O(N \log N)$ time, $O(N \log N)$ disk, $O(\log N)$ memory.

Usage in R. We implemented the disk-based version of Algorithm 1 in C++ code with an interface in the R package **PeakSegDisk**. To illustrate its usage we first load a set of genomic data,

```
> library(PeakSegDisk)
> data(Mono27ac, package="PeakSegDisk")
> Mono27ac$coverage
```

```
      chrom chromStart chromEnd count
   1: chr11      60000   132601     0
   2: chr11     132601   132643     1
   3: chr11     132643   146765     0
   4: chr11     146765   146807     1
   5: chr11     146807   175254     0
  ---
6917: chr11     579752   579792     1
6918: chr11     579792   579794     2
6919: chr11     579794   579834     1
6920: chr11     579834   579980     0
6921: chr11     579980   580000     1
```

```
>
```

Note that the 4 column bedGraph format shown above must be used to represent a data set. Futhermore a run-length encoding should be used for data sets that have runs of the same values. Each row represents a sequence of identical data values. For example the first row means that the value 0 occurs on the 72601 positions in [60000,132601), the second row means a value of 1 for the 42 positions in [132601,132643), etc. This run-length encoding results in significant savings in disk space and time [Cleynen et al., 2014]; for example in the data set above there are only 6921 lines used to represent 520000 data values.

In order to handle very large data sets while using only $O(\log N)$ memory, the algorithm reads input data from a text file on disk (and never actually stores the entire data set in memory). So before using the algorithm we must save the data set to disk, in bedGraph format (the four columns shown above, separated by tabs). Note that the file name must be `coverage.bedGraph`:

```
> data.dir <- file.path("Mono27ac", "chr11:60000-580000")
> dir.create(data.dir, showWarnings=FALSE, recursive=TRUE)
> write.table(
+   Mono27ac$coverage, file.path(data.dir, "coverage.bedGraph"),
+   col.names=FALSE, row.names=FALSE, quote=FALSE, sep="\t")
>
```

After saving the file to disk, we can run the algorithm using the code below:

```
> ## Compute one model with penalty=10000
> fit <- PeakSegDisk::problem.PeakSegFPOP(data.dir, "10000")
>
```

For the first argument you must give the folder name (not the `coverage.bedGraph` file name) to the problem.PeakSegFPOP function. Note that the second argument must be a character string that represents a penalty value (non-negative real number, larger penalties yield fewer peaks). The smallest value is `"0"` which yields max peaks, and the largest value is `"Inf"` which yields no peaks. It must be an R character string (not a real number) because that string is used to create files which are used to store/cache the results. If the files already exist (and are consistent) then problem.PeakSegFPOP just reads them; otherwise it runs the dynamic programming C++ code in order to create those files.

The returned `fit` object is a named list of data.tables. The `fit$loss` component shown below is one row that contains general information about the computed model:

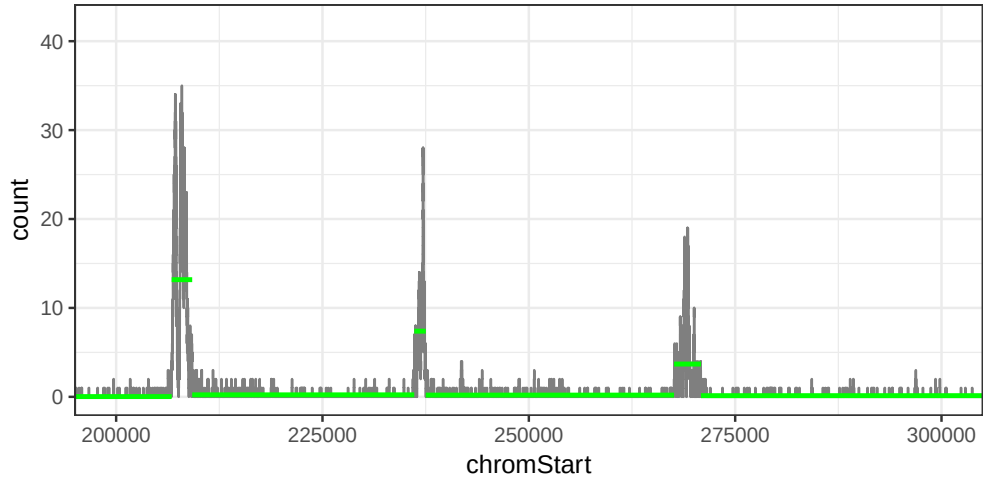
```
> fit$loss

   penalty segments peaks  bases mean.pen.cost total.loss equality.constraints
1:   10000       15     7 520000     0.2189332   43845.26                   0
   mean.intervals max.intervals
1:       14.42581            41

>
```

Above we can see that the optimal model for $\lambda = 10^4$ had $P = 7$ peaks ($K = 15$ segments/$K - 1 = 14$ changepoints). The `fit$segments` component is used to visualize three of those peaks in a subset of the data below.

```
> library(ggplot2)
> gg <- ggplot()+theme_bw()+
+   geom_step(aes(chromStart, count), color="grey50", data=Mono27ac$coverage)+
+   geom_segment(aes(chromStart, mean, xend=chromEnd, yend=mean),
+     color="green", size=1, data=fit$segments)+
+   coord_cartesian(xlim=c(2e5, 3e5))
> print(gg)
>
```



3.2 Sequential search algorithm for P^* peaks

Note that in GFPOP (Algorithm 1), the user inputs a penalty λ, and is unable to directly choose the number of segments/peaks. In this section, we propose an algorithm that allows the user to specify the number of peaks. The algorithm then repeatedly calls GFPOP until it finds the most likely model with at most the specified number of peaks.

To understand how the algorithm works, we must review the relationship between the Optimal Partitioning and Segment Neighborhood problems [Maidstone et al., 2016]. We define the optimal loss for a given number of peaks P as

$$L_P = \min_{\substack{\mathbf{m}\in\mathbb{R}^N,\ \mathbf{s}\in\{0,1\}^N \\ \mathbf{c}\in\{-1,0,1\}^{N-1}}} \sum_{i=1}^{N} \ell(m_i, z_i) \tag{15}$$

$$\begin{aligned} \text{subject to} \quad & c_i = 0 \Rightarrow m_i = m_{i+1} \text{ and } s_i = s_{i+1}, \\ & c_i = 1 \Rightarrow m_i \leq m_{i+1} \text{ and } (s_i, s_{i+1}) = (0,1), \\ & c_i = -1 \Rightarrow m_i \geq m_{i+1} \text{ and } (s_i, s_{i+1}) = (1,0), \\ & s_1 = s_N = 0, \\ & P = \sum_{i=1}^{N-1} I(c_i = 1). \end{aligned} \tag{16}$$

The problem above is the Segment Neighborhood version of the Optimal Partitioning problem that GFPOP solves (5). The penalty λ is absent, and the model complexity (the number of peaks) has moved to a constraint (16). Recall that $F(\lambda)$ is the minimum value of the Optimal Partitioning problem (5). It can be written in terms of the solution to the Segment Neighborhood problem (15),

$$F(\lambda) = \min_{P\in\{0,1,\ldots,P_{\max}\}} L_P + \lambda P. \tag{17}$$

The equation above makes it clear that there are only a finite number of optimal changepoint models (from 0 to $P_{\max}$ peaks). $F(\lambda)$ is a concave, non-decreasing function that can be computed as the minimum of a finite number of affine functions $f_P(\lambda) = L_P + \lambda P$.

We now assume the user wants to compute the optimal model with a fixed number of peaks P^*. To compute that model we will maximize the function

$$G(\lambda) = F(\lambda) - P^*\lambda = \min_{P\in\{0,1,\ldots,P_{\max}\}} \underbrace{L_P + \lambda(P - P^*)}_{g_P(\lambda)}. \tag{18}$$

From the equation above it is clear that $G(\lambda)$ is a concave function that can be computed as the minimum of a finite number of affine functions $g_P(\lambda) = L_P + \lambda(P - P^*)$. For an example G function see Figure 3.

Discussion of pseudocode. Algorithm 2 summarizes the sequential search. The main idea of the sequential search algorithm is to keep track of a lower bound $\underline{p} < P^*$ and upper bound $\overline{p} > P^*$ on the number of peaks computed thus far. The algorithm starts with $\lambda = 0$, $\overline{p} = P_{\max}$ (line 2) and $\lambda = \infty, \underline{p} = 0$ (line 3). At each iteration of the algorithm, we find the intersection of the affine functions $g_{\underline{p}}(\lambda) = g_{\overline{p}}(\lambda)$, which leads to a new candidate penalty $\lambda_{\text{new}} = (L_{\overline{p}} - L_{\underline{p}})/(\underline{p} - \overline{p})$ (line 5). As previously described [Haynes et al., 2017], there are two possibilities for the solution to the Optimal Partitioning problem:

- GFPOP(λ_{new}) yields $\underline{p}$ or $\overline{p}$ peaks (line 7). In that case $\max_\lambda G(\lambda) = G(\lambda_{\text{new}}) = g_{\underline{p}}(\lambda_{\text{new}}) = g_{\overline{p}}(\lambda_{\text{new}})$ and there is no Optimal Partitioning model with P^* peaks. We terminate the algorithm by returning the model with $\underline{p}$ peaks.
- GFPOP(λ_{new}) yields a new model with p_{new} peaks. If $p_{\text{new}} = P^*$ then $\max_\lambda G(\lambda) = L_{P^*}$ and we return this model (line 8). Otherwise it must be true that $\underline{p} < p_{\text{new}} < \overline{p}$. If $\underline{p} < p_{\text{new}} < P^*$ then we use p_{new} for a new lower bound $\underline{p}$ (line 9); otherwise we use it for a new upper bound $\overline{p}$ (line 10).

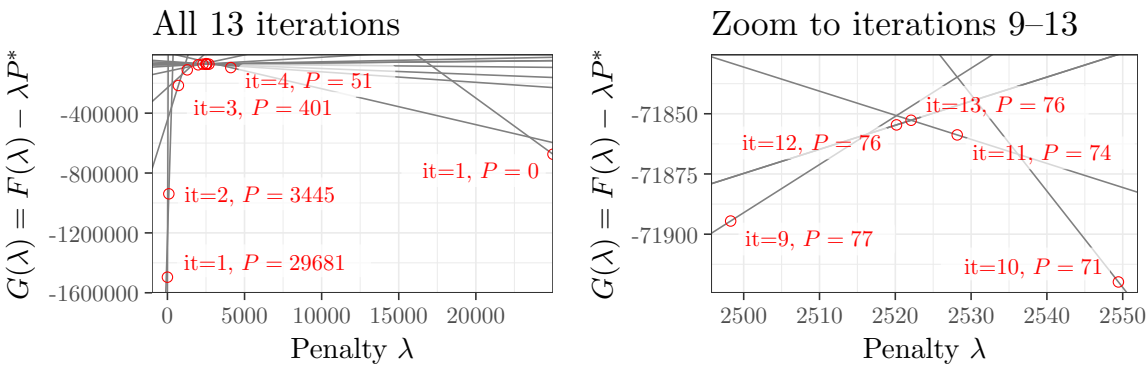


Figure 3: Example of a $G(\lambda)$ function which is maximized in order to find the most likely model with at most $P^* = 75$ peaks. Red dots show $G(\lambda)$ values evaluated by the algorithm; grey lines show affine functions $g_P(\lambda) = L_P + (P - P^*)\lambda$ used to determine the next λ value (line 5 of Algorithm 2). **Left:** iteration 1 runs GFPOP with $\lambda \in \{0, \infty\}$, resulting in initial lower bound of $\underline{p} = 0$ peaks and upper bound of $\overline{p} = 29681$ peaks. In iteration 2 the algorithm finds the intersection of the upper/lower bound lines $g_0(\lambda) = g_{29681}(\lambda)$ at $\lambda = 90.9$; running GFPOP with that penalty reduces the upper bound to $\underline{p} = 3445$. **Right:** In the last iteration (13), we run GFPOP with $\lambda = 2522.1$ (which is where g_{74} intersects g_{76}), resulting in 76 peaks when we already have $\overline{p} = 76$ as an upper bound (computed in iteration 12). The maximum of G is thus $G(2522.1) = g_{74}(2522.1) = g_{76}(2522.1)$; the algorithm returns the model with $P = 74$ peaks.

Algorithm 2 Sequential search for P^* peaks using GFPOP.

1: Input: data $\mathbf{z} \in \mathbb{R}^N$, target peaks P^*.
2: $\overline{L}, \overline{p} \leftarrow \text{GFPOP}(\mathbf{z}, \lambda = 0)$ // initialize upper bound to max peak model
3: $\underline{L}, \underline{p} \leftarrow \text{GFPOP}(\mathbf{z}, \lambda = \infty)$ // initialize lower bound to 0 peak model
4: While $P^* \notin \{\underline{p}, \overline{p}\}$:
5: $\quad \lambda_{\text{new}} = (\overline{L} - \underline{L})/(\underline{p} - \overline{p})$
6: $\quad L_{\text{new}}, p_{\text{new}} \leftarrow \text{GFPOP}(\mathbf{z}, \lambda_{\text{new}})$
7: $\quad$ If $p_{\text{new}} \in \{\underline{p}, \overline{p}\}$: return model with $\underline{p}$ peaks.
8: $\quad$ If $p_{\text{new}} = P^*$: return model with p_{new} peaks.
9: $\quad$ If $p_{\text{new}} < P^*$: $\underline{L}, \underline{p} \leftarrow L_{\text{new}}, p_{\text{new}}$ // new lower bound
10: $\quad$ Else: $\overline{L}, \overline{p} \leftarrow L_{\text{new}}, p_{\text{new}}$ // new upper bound

Computational complexity. The space complexity is the same as GFPOP: $O(N \log N)$ disk and $O(\log N)$ memory. Its time complexity is linear in the number of iterations of the while loop (line 4). Empirically we see $O(\log P^*)$ iterations (Section 4.4), which implies an overall time complexity of $O(N \log(N) \log(P^*))$.

Usage in R. The R code below computes the optimal model with 17 peaks:

```
> ## Compute the optimal model with 17 peaks.
> fit <- PeakSegDisk::problem.sequentialSearch(data.dir, 17L)
>
```

If you want to see how many iterations/penalties the algorithm required in order to compute the optimal model with 17 peaks, you can look at the `fit$others` component:

```
> fit$others[, list(iteration, under, over, penalty, peaks, total.loss)]

   iteration under over   penalty peaks  total.loss
1:         1    NA   NA    0.0000  3199 -130227.291
```

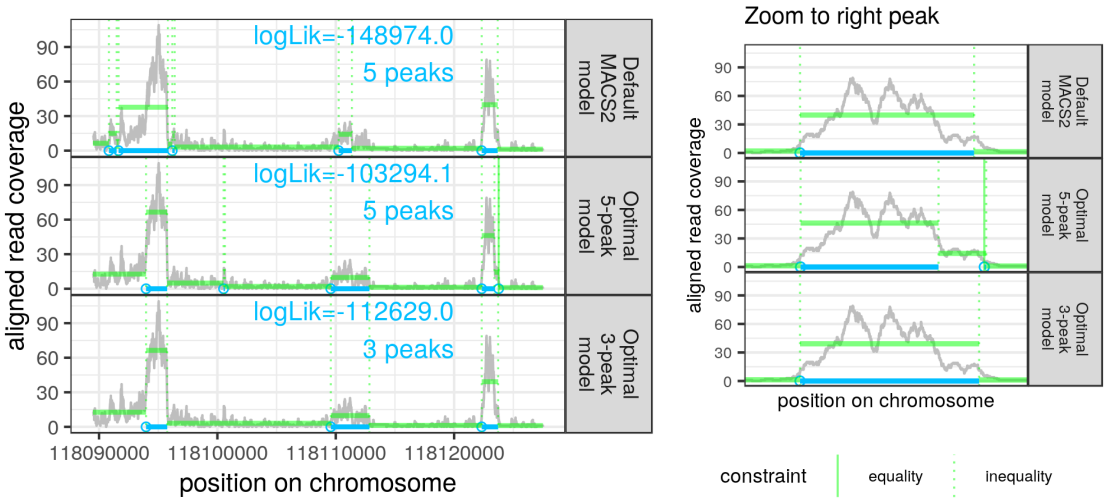


Figure 4: One ChIP-seq data set with three peak models. (green horizontal segment means; green dotted vertical lines for changepoints; blue bars for peaks; blue dots for peak starts) **Top:** the MACS2 algorithm (a heuristic from the bioinformatics literature) computed a sub-optimal model with five peaks for these data. **Middle:** the most likely model with five peaks contains one equality constraint between segment means (see zoomed figure on the right), which suggests that there are less than five easily interpretable peaks. **Bottom:** the most likely model with three peaks is also more likely than the MACS2 model.

```
2:         1    NA    NA       Inf     0  375197.873
3:         2     0  3199  157.9947   224  -62199.931
4:         3     0   224 1952.6688    17    2640.128

>
```

The output above shows that the algorithm only used three iterations to compute the optimal model with 17 peaks. The `under` and `over` columns show the current values of $\underline{p}$ and $\overline{p}$, respectively. The `peaks` and `total.loss` are $p_{\text{new}}, L_{\text{new}}$ from the model that resulted from running GFPOP with $\lambda =$ `penalty`. Note that iteration 1 evaluates both extreme penalties $\lambda \in \{0, \infty\}$ in parallel (and $\lambda = \infty$ is the trivial model with 0 peaks that can be computed without dynamic programming), so these two models are considered a single iteration.

4 Results on genomic data

In this section we discuss several applications of our algorithms in some typical genomic data sets. We downloaded the `chipseq` data set from the UCI machine learning repository [Newman and Merz, 1998]. We considered 4951 data sets ranging in size from $N = 10^3$ to $N = 10^7$ data points to segment (lines in the bedGraph file).

4.1 Application: Computing the maximum likelihood model with a given number of peaks

In this section we show that our algorithms can be used to compute the most likely model for a given number of peaks. A subset of one data set is shown in Figure 4, along with three segmentation/peak models. In the top panel, we show the peak model that results from running MACS2, a heuristic algorithm from the

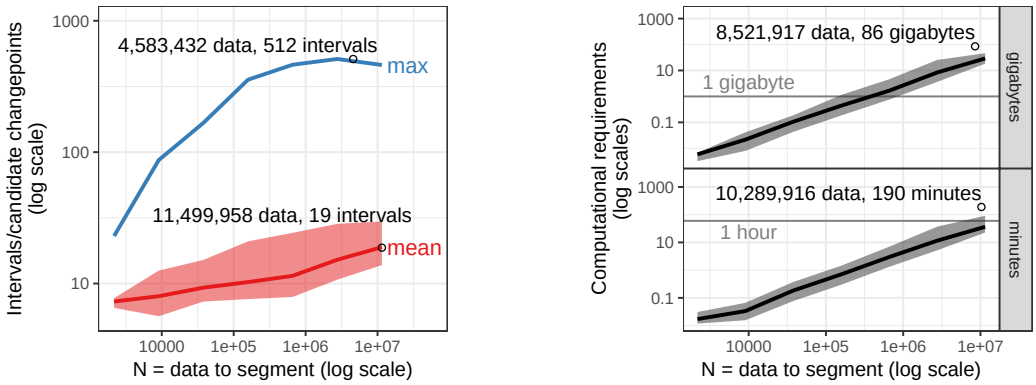


Figure 5: In our empirical tests, the computational requirements of the GFPOP algorithm were log-linear $O(N \log N)$ in the number of data points N to segment. **Left:** we analyzed the number of intervals I (candidate changepoints) stored in the $C_t(\mu)$ cost functions, because the total time/space complexity is $O(NI)$. We observed empirically that the mean number of intervals $I = O(\log N)$ (red curve). Even the maximum number of intervals (blue curve) is much less than N. **Right:** storage on disk (top panel) and computation time (bottom panel) are empirically $O(N \log N)$. Error bands show median and 5%/95% quantiles over several data sets of a given size N; black dots and text show computational requirements for the most extreme data sets.

bioinformatics literature [Zhang et al., 2008]. It detects five peaks, so we ran Algorithm 2 with $P^* = 5$ on these data in order to compute the most likely model with at most 5 peaks (shown in middle panel). It is clear that the optimal 5 peak model is a better fit in terms of likelihood (as expected); it is also a better fit visually, especially for the peak on the left. Furthermore the optimal 5 peak model actually has one equality constraint between adjacent segment means, suggesting that there are less than five easily interpretable peaks. Therefore we also computed the optimal 3 peak model (bottom panel), which also has a higher log-likelihood than the 5 peak MACS2 model. Overall it is clear that our algorithms can be used to compute models which are both more likely and simpler (with fewer peaks) than heuristics such as MACS2.

4.2 GFPOP is empirically log-linear

To measure the empirical time complexity of GFPOP (Algorithm 1), we ran it on all 4951 genomic data sets, with a grid of penalty values $\lambda \in (\log N, N)$ for each data set of size N. The overall theoretical time/space complexity is $O(NI)$, where I is the number of intervals (candidate changepoints) stored in the $C_{s,t}$ optimal cost functions. During each run we therefore recorded the mean and max number of intervals over all s, t. We observed that the empirical mean/max number of intervals increases logarithmically with data set size, $I = O(\log N)$ (Figure 5, left). Remarkably, for the largest data set ($N = 11,499,958$) the algorithm only computed a mean of $I = 19$ intervals. The most intervals computed to represent any single $C_{s,t}$ function was 512 intervals for one data set with $N = 4,583,432$.

Since empirically $I = O(\log N)$ in these genomic data sets, we expected an overall time/space complexity of $O(N \log N)$. The empirical measurements of time and space requirements are consistent with this expectation (Figure 5, right). For the largest data sets ($N = 10^7$), the algorithm takes only about 80 gigabytes of storage and 1 hour of computation time. Overall these results suggest that GFPOP can be used to compute optimal peak models for genomic data in $O(N \log N)$ space/time.

4.3 Disk storage is slower than memory by a constant factor

In the previous section, we discussed how tens of gigabytes of storage are required to run GFPOP when $N = 10^7$. Since typical computers may not have enough memory, our implementation uses disk-based

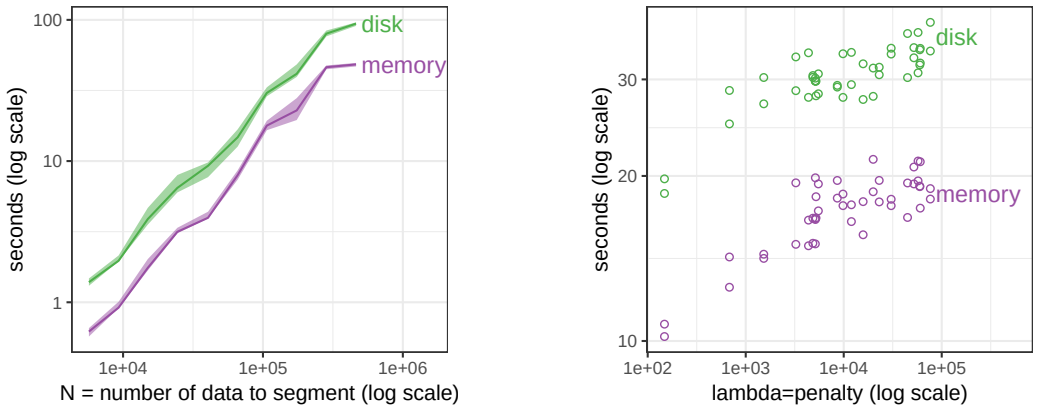


Figure 6: The disk-based storage method is only a constant factor slower than the memory-based method. We benchmarked both methods on several small data sets ($N \leq 462,890$) for which optimal models could be computed using 1GB of storage. **Left:** computation time is empirically $O(N \log N)$ for both storage methods, but the disk-based method is slower by a constant factor. Median line and quartile band computed over several penalty values for a given data set. **Right:** fixing one data set data set with $N = 106,569$, the computation time increases with penalty value λ for both storage methods.

storage. We compared our disk-based implementation to another memory-based implementation, in terms of computation time on small data sets for which GFPOP uses $<$ 1GB of storage. We observed that disk storage is slower than memory storage by a constant factor (1.7–2.3$\times$, Figure 6), which was expected.

4.4 Sequential search is faster than Segment Neighborhood

In this section we compare the number of $O(N \log N)$ dynamic programming iterations required for the proposed sequential search (Algorithm 2) and the previous Generalized Pruned Dynamic Programming Algorithm (GPDPA) of Hocking et al. [2017]. Both algorithms compute the solution to the Segment Neighborhood problem (optimal model with at most P peaks). The GPDPA requires exactly $2P$ iterations of dynamic programming, each of which is an $O(N \log N)$ operation. In contrast, the proposed sequential search (Algorithm 2) needs to solve for a sequence of penalties, each of which is done via GFPOP in $O(N \log N)$ time.

For two data sets with $N \approx 10^6$ we therefore recorded the empirical number of times GFPOP was called by the sequential search algorithm. We observed that the number of GFPOP calls grows logarithmically with P (Figure 7, left). For a large number of peaks ($P > 5$), it is clearly faster to use the sequential search algorithm (Figure 7, right). Overall these experiments indicate that the time complexity of the sequential search in genomic data is $O(N \log(N) \log(P))$ for N data and P peaks.

4.5 Application: computing a zero-error peak model

In this section we study the perfomance of the proposed algorithms in a typical application. In the UCI `chipseq` data set, there are labels that indicate subsets of the data with or without peaks. In this context the labels can be used to compute false positive and false negative rates for any peak model. For example Figure 8 shows one data set with six labels and four peak models computed via GFPOP. Small penalties result in too many peaks, and large false positive rates. Large penalties result in too few peaks, and large false negative rates. A range of intermediate penalties/peaks achieves zero label errors. The labels can thus be used to determine an appropriate number of peaks (with zero errors) for each data set.

More generally, after computing GFPOP models for a range of penalties for each data set, we computed the label error of each model. For each data set we computed the min/max peaks that achieves zero label

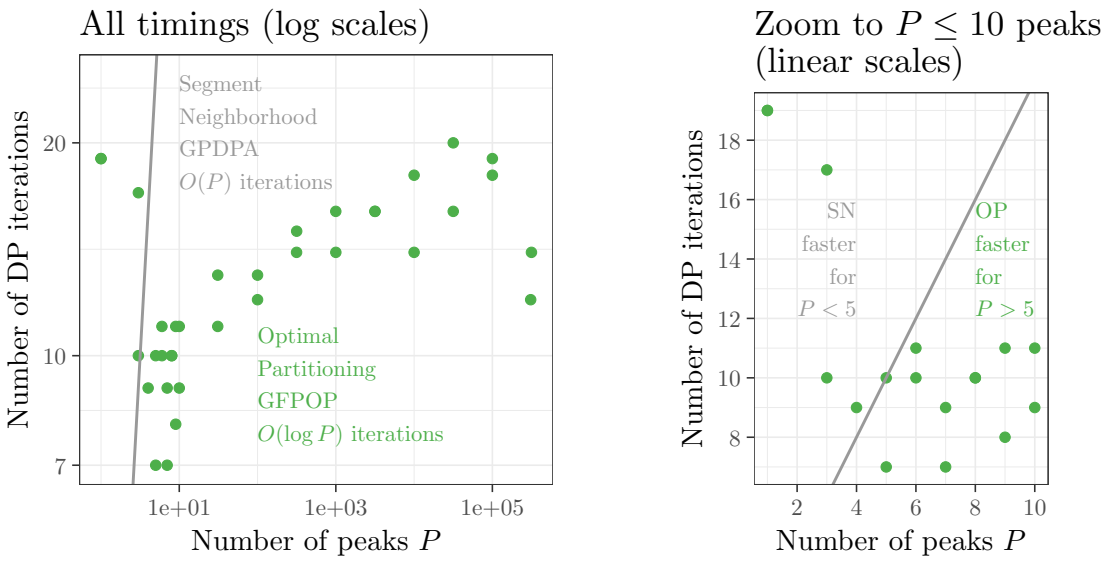


Figure 7: Comparison of time to compute optimal model with at most P peaks using Segment Neighborhood (grey lines) and Optimal Partitioning with proposed sequential search (green dots). GFPOP with sequential search (Algorithm 2) was used to compute optimal models with different numbers of peaks P, for two data sets with $N \approx 10^6$. **Left:** the number of iterations is linear $O(P)$ for Segment Neighborhood (grey line) but empirically $O(\log P)$ for Optimal Partitioning with sequential search (green dots). **Right:** Optimal Partitioning is empirically faster for computing models with $P > 5$ peaks (10 segments); Segment Neighborhood is faster for smaller models.

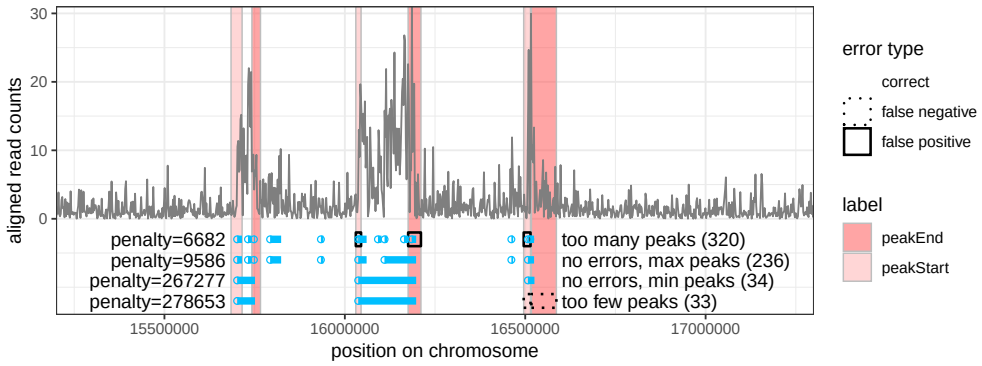


Figure 8: Labels are used to compute an error rate for each peak model (blue bars), defined as the sum of false positive and false negative labels (rectangles with black outline). This H3K36me3 ChIP-seq data set has $N = 1,254,751$ data to segment on a subset of chr12, but in the plot above we show only the 82,233 data (grey signal) in the region around the labels (colored rectangles). The model with penalty=6682 results in 320 peaks, which is too many (three false positive labels with more than one peak start/end). Conversely, the model with penalty=278653 results in 33 peaks, which is too few (only two peaks in the plotted region, resulting in two false negative labels on the right where there should be exactly one peak start/end). The range of penalties between 9586 and 267277 results in models with between 34 and 236 peaks, and achieves zero label errors.

errors (34/236 in Figure 8), along with the mean of those two values, $(34 + 236)/2 = 135$. We plot the mean number of peaks that achieves zero label errors as a function of data set size in Figure 9. In these data it is

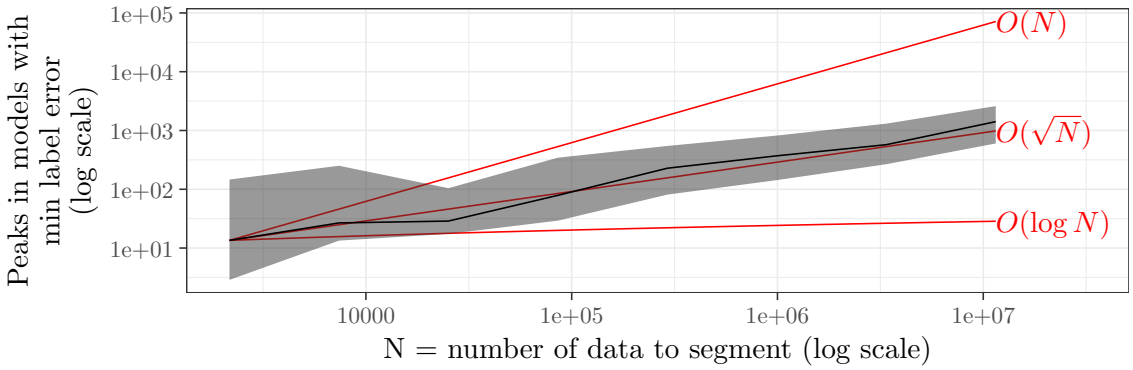


Figure 9: The model with minimal label errors has $O(\sqrt{N})$ peaks in a data set of size N. For each data set we computed peak models with minimal label errors (see Figure 8); we then plot the number of peaks in minimal error models as a function of data set size N. Black median line and grey quartile band computed over several data sets of a given size N; asymptotic reference lines shown in red.

clear that models with $O(\sqrt{N})$ peaks achieve zero label errors.

Computing the optimal model with $O(\sqrt{N})$ peaks is computationally expensive using the Segment Neighborhood algorithm (**PeakSegOptimal** package), because the overall complexity would be $O(N\sqrt{N}\log N)$. For example in $N = 10^7$ data, $P = O(\sqrt{N}) = 1414$ peaks achieves zero label errors. Computing the optimal model with Segment Neighborhood would thus require 2828 $O(N \log N)$ DP iterations. If we assume that each iteration would have similar computational requirements as one $O(N \log N)$ run of GFPOP, each would require about 1 hour and 80 gigabytes (Figure 5). Overall that would mean 220 terabytes of storage and 17 weeks of computation time, which is much too expensive in practice.

Instead, we can use the proposed sequential search (Algorithm 2) to compute a zero-error model with $O(\sqrt{N})$ peaks. In our empirical tests, we observed that only $O(\log N)$ GFPOP calls are required to compute $O(\sqrt{N})$ peaks (Figure 10, left). In particular for $N = 10^7$ data only 10–15 GFPOP calls are required, which is significantly fewer than the 2828 DP iterations that would be required for the Segment Neighborhood solver in the **PeakSegOptimal** package.

We also observed that the empirical timings of the sequential search are only a log-factor slower than solving for one penalty (Figure 10, right). In particular for $N = 10^7$ data the sequential search takes on the order of hours, which is much less than the weeks that would be required to solve the Segment Neighborhood problem. Overall these empirical results indicate that the sequential search algorithm in the **PeakSegDisk** package can be used to compute a model with $O(\sqrt{N})$ peaks in $O(N(\log N)^2)$ time.

5 Summary and discussion

This paper presented two new algorithms for constrained optimal changepoint detection. We presented Generalized Functional Partitioning Optimal Partitioning (GFPOP) which computes the optimal model for one penalty λ. We also proposed a sequential search algorithm which repeated calls GFPOP in order to compute the most likely model with at most P peaks.

We analyzed the proposed algorithms by running them on a set of genomic data sets ranging from $N = 10^3$ to $N = 10^7$. First, we showed that the algorithms can be used to compute models which are more likely than existing heuristics, and often simpler (fewer peaks).

Second, we studied the empirical complexity of GFPOP as the a function of the number of data N. We showed that GFPOP requires $O(N \log N)$ time, $O(N \log N)$ space, and $O(\log N)$ memory. We furthermore showed that using disk-based storage is only a constant factor slower than memory-based storage. Overall we showed that GFPOP can be used to compute optimal peak models for up to $N = 10^7$ data in reasonable

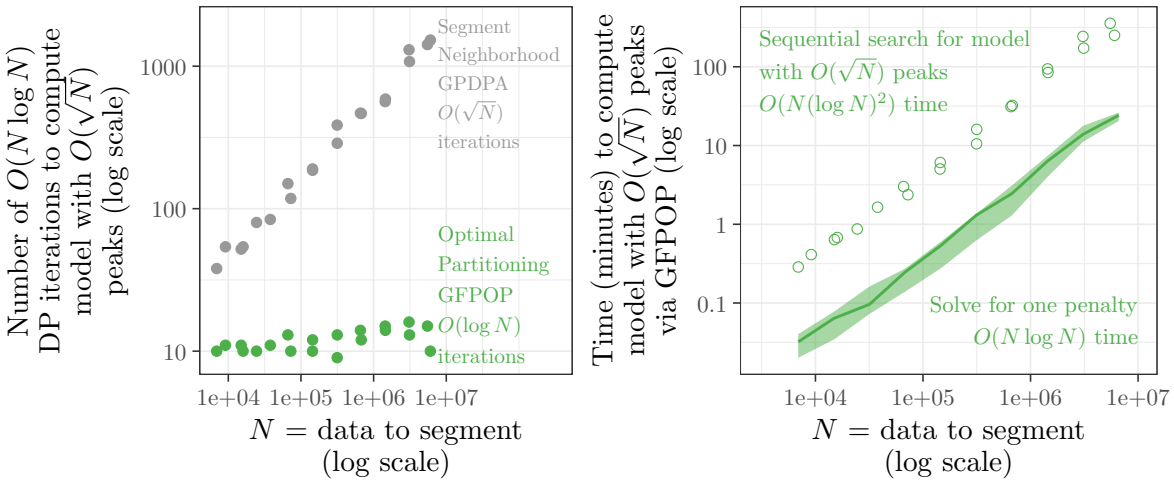


Figure 10: Computing a zero-error model with $O(\sqrt{N})$ peaks is possible in $O(N(\log N)^2)$ time using our proposed Optimal Partitioning Search algorithm. **Left:** Segment Neighborhood requires $O(\sqrt{N})$ dynamic programming iterations to compute a model with $O(\sqrt{N})$ peaks; our proposed Optimal Partitioning search algorithm requires only $O(\log N)$ iterations. **Right:** Optimal Partitioning solves for one penalty in $O(N \log N)$ space/time (median line and 5%/95% quantile band over data sets and penalties); finding the zero-error model with $O(\sqrt{N})$ peaks takes $O(N(\log N)^2)$ time/space – only a log factor more (points).

amounts of time (minutes).

Third, we studied the empirical complexity of sequential search as a function of data size N and number of peaks P. We showed that it requires $O(N \log(N) \log(P))$ time, $O(N \log N)$ disk, $O(\log N)$ memory. In particular we showed that it is always faster than Segment Neighborhood solvers for models with $P > 5$ peaks.

Finally we analyzed the labels in our benchmark of genomic data, which indicated that an appropriate number of peaks is $P = O(\sqrt{N})$. We showed that sequential search computes the model with $O(\sqrt{N})$ peaks in $O(N(\log N)^2)$ time, whereas existing Segment Neighborhood algorithms would be $O(N\sqrt{N} \log N)$. We showed that for $N = 10^7$ data our approach only requires hours/gigabytes of time/space to compute optimal models. Our algorithms thus make it practical for the first time to compute optimal models with many peaks for genomic data sets.

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`

References

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

A. Barski, S. Cuddapah, K. Cui, T.-Y. Roh, D. E. 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.

J. D. Buenrostro, B. Wu, H. Y. Chang, and W. J. Greenleaf. ATAC-seq: A Method for Assaying Chromatin Accessibility Genome-Wide. *Current Protocols in Molecular Biology*, 2015.

H. Choi, A. I. Nesvizhskii, D. Ghosh, and Z. S. Qin. Hierarchical hidden markov model with application to joint analysis of chip-chip and chip-seq data. *Bioinformatics*, 25(14):1715–1721, 2009.

A. Cleynen, M. Koskas, E. Lebarbier, G. Rigaill, and S. Robin. Segmentor3IsBack: an R package for the fast and exact segmentation of Seq-data. *Algorithms for Molecular Biology*, 9:6, 2014.

K. Frick, A. Munk, and H. Sieling. Multiscale change point inference. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(3):495–580, 2014.

P. Fryzlewicz. Wild binary segmentation for multiple change-point detection. 42, 11 2014.

K. Haynes, I. A. Eckley, and P. Fearnhead. Computationally efficient changepoint detection for a range of penalties. *Journal of Computational and Graphical Statistics*, 26(1):134–143, 2 2017. ISSN 1061-8600.

T. Hocking, G. Rigaill, P. Fearnhead, and G. Bourque. A log-linear time algorithm for constrained change-point detection. arXiv:1703.03352, 2017.

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

T. D. 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.

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

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

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

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

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

C. B. D. Newman and C. Merz. UCI repository of machine learning databases, 1998.

M. Pierre-Jean, G. Rigaill, and P. Neuvial. Performance evaluation of DNA copy number segmentation methods. *Briefings in Bioinformatics*, 2015.

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 and T. D. Hocking. *fpop: Segmentation using Optimal Partitioning and Function Pruning*, 2016. URL `https://R-Forge.R-project.org/projects/opfp/`. R package version 2016.10.25/r55.

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