Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

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

Toby Dylan Hocking Northern Arizona University

Guillem Rigaill INRAE

Paul Fearnhead Lancaster University

Guillaume Bourque McGill University

Abstract

We describe a new algorithm and R package for peak detection in genomic data sets using constrained changepoint models. 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 a dynamic programming 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 analyzing 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.

Keywords: Dynamic programming, optimal changepoint detection, peak detection, genomic data, R.

1. Introduction

This paper presents a new algorithm, and corresponding implementation as R package **PeakSegDisk**, for detecting an optimal sequence of up/down changes in large count data, which are common in genomics. The algorithm is based on the technique of *functional pruning* (Maidstone, Hocking, Rigaill, and Fearnhead 2016), which represents the cost as a function of the segment mean parameter, and reduces the number of candidate changepoints while guaranteeing optimality. Note that throughout this paper the term "optimal" will be used to describe algorithms that compute an exact solution to some well-defined optimization problem (e.g., maximum likelihood or minimum cost). For models that are defined in terms of maximizing a likelihood function, we use the terms "more likely" and "most likely" to compare likelihood values for specific models/algorithms.

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, Cuddapah, Cui, Roh, Schones, Wang, Wei, Chepelev, and Zhao 2007); ATAC-seq data measure open chromatin (Buenrostro, Wu, Chang, and Greenleaf 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. These long contiguous runs with large counts are referred to as "peaks" and they typically correspond to active regions of the genome in specific samples and cell types. An important sub-problem in analyzing these data is identifying the precise genomic positions of the starts and ends of these peak regions. Each peak is separated from other peaks by background/noise regions (i.e., it is impossible for two peaks to be directly adjacent to one another without intervening background noise). This is a central assumption of all peak detection algorithms (Wilibanks and Facciotti 2010); in practice this means that if there are two or more adjacent regions that result in peaks, any peak detection algorithm will report them as a single peak.

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, Rigaill, Fearnhead, and Bourque (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, Nesvizhskii, Ghosh, and Qin 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 between optimal changepoint models and other algorithms we refer the reader to the previous work of Hocking, Goerner-Potvin, Morin, Shao, Pastinen, and Bourque (2016). We focus the remainder of the paper on optimal changepoint models, i.e., penalized maximum

likelihood approaches (and their exact computation).

1.2. Models with no constraints between adjacent segment means

Denote a sequence of N data by $z_1, \ldots, z_N$. We assume the data sequence 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 neighboring 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 let m_k denote the segment-specific model parameter for each segment 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 each 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 a 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 "Segment Neighborhood" 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 (Bellman 1961). The first application of this principle to changepoint detection was the Segment Neighborhood algorithm, which computes optimal segmentations from 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 such as the Pruned Dynamic Programming Algorithm (PDPA) of Rigaill (2015), which empirically take $O(KN \log N)$ time. The novelty of this algorithm 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** was orphaned and had to be 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, Rigaill, and Neuvial 2015). Cleynen and Lebarbier (2014) described an implementation 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 in advance how many segments K are present in the data. To estimate K it is common to use information criteria that take into 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, although others exist (Zhang and Siegmund 2007; Cleynen and Lebarbier 2014; Arlot, Brault, Baudry, Maugis, and Michel 2016). The $O(N^2)$ dynamic programming algorithm proposed by Jackson, Scargle, Barnes, Arabhi, Alt, Gioumousis, Gwin, Sangtrakulcharoen, Tan, and Tsai (2005) solves the corresponding 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 in the optimization objective 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). There are several existing algorithms for choosing the penalty parameter λ, and this is an active area of ongoing research. There are several classical unsupervised heuristics such as the Bayesian Information Criterion (BIC) (Schwarz 1978; Yao 1988) as well as modern modifications for genomic data sets (Lebarbier 2005; Zhang and Siegmund 2007). There are also supervised algorithms which are useful when there are some data sets with labels that indicate presence/absence of changepoints in specific regions (Rigaill, Hocking, Vert, and Bach 2013).

There has been substantial research into faster algorithms for solving the Optimal Partitioning problem. The Pruned Exact Linear Time (PELT) algorithm (`changepoint::cpt.mean` in R) is $O(N)$ time if K increases linearly with N (Killick, Fearnhead, and Eckley 2012; Killick and Eckley 2014). The Functional Pruning Optimal Partitioning (FPOP) algorithm (`fpop::Fpop` in R) has been shown to be empirically $O(N \log N)$ and faster than PELT in practice for several data sets (Maidstone *et al.* 2016). In particular, the number of changepoints considered by FPOP is always smaller than (or equal to) the number of changepoints considered by PELT, so when PELT is $O(N)$ FPOP is as well. See Table 1 for a summary of the different dynamic programming algorithms and implementations.

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 (Maidstone *et al.* 2016). Several of these alternative algorithms are available in R. For example, the **wbs** package implements a stochastic algorithm called wild binary segmentation (Fryzlewicz 2014; Baranowski and Fryzlewicz 2019). Efficient implementations of the classical binary segmentation heuristic are available in R as `fpop::multiBinSeg` and `binseg::binseg_normal` (Hocking 2019). The **stepR** and **FDRSeg** packages implement a

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.

multiscale approach to changepoint inference, and have linear run times if the number of changes increases linearly in the number of observations (Frick, Munk, and Sieling 2014; Li, Munk, and Sieling 2016; Li and Sieling 2017; Pein, Hotz, Sieling, and Aspelmeier 2019).

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, Rigaill, and Bourque (2015) introduced an $O(KN^2)$ 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. Note that in this constrained model, peaks can neither be predicted at the beginning nor at the end of the data. This is a sensible constraint in genomic data, because peaks are not expected to occur near the beginning/end of the data. 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 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 2018).

1.4. Contributions

This paper presents two new efficient algorithms for constrained optimal changepoint detec-

	No changepoint pruning	Functional pruning
Segment Neighborhood K segments	Constrained DPA Sub-optimal, $O(KN^2)$ Hocking *et al.* (2015) **PeakSegDP**	GPDPA Optimal, $O(KN \log N)$ Hocking *et al.* (2017) **PeakSegOptimal**
Optimal Partitioning Penalty λ		GFPOP 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.

tion (Section 3). Whereas previous work has focused on the unconstrained Optimal Partitioning problem (Table 1) and the constrained Segment Neighborhood problem (Table 2), this paper is the first to propose a solver for the constrained Optimal Partitioning problem. We also propose a sequential search that uses the Optimal Partitioning algorithm as a subroutine for solving the Segment Neighborhood problem. These algorithms are implemented using efficient disk-based storage in the R package **PeakSegDisk**. Our results include a detailed analysis of the empirical time/space complexity of these algorithms in a benchmark of genomic data (Section 4). The paper concludes with a discussion of the significance of these contributions in the context of related work (Section 5).

2. Statistical models and optimization problems

In this section we first introduce an equivalent formulation of the Optimal Partitioning problem, then add constraints on adjacent segment means.

2.1. Unconstrained Optimal Partitioning problem

Recall that $z_1, \ldots, z_N \in \mathbb{Z}_+$ is a sequence of N non-negative count data. Maximizing the Poisson likelihood of a data point z given a mean parameter m is equivalent to minimizing the Poisson loss, $\ell(m, z) = m - z \log m$. Below we write an equivalent version of the Optimal Partitioning problem (2), 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.

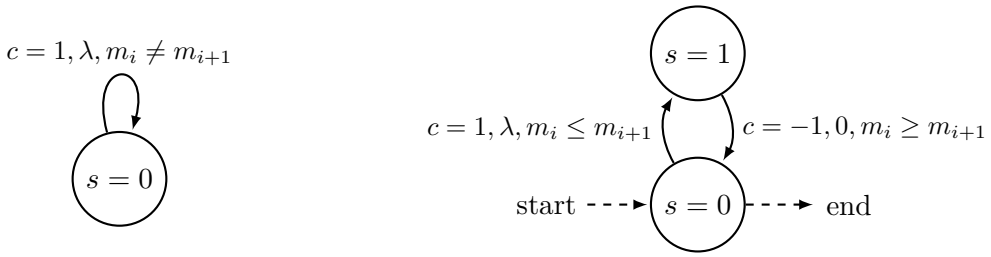


Figure 1: State graphs for two changepoint models. Nodes represent states and solid edges represent changepoints. **Left:** one-state model in which each change $c = 1$ has a penalty of λ, problem (3). **Right**: two-state model with up-down constraints between adjacent segment means, problem (5). State $s = 0$ represents background noise (small values) whereas state $s = 1$ represents peaks (large values). Constraint $c = 1$ enforces a non-decreasing change with a penalty of λ; $c = -1$ enforces a non-increasing change with a penalty of 0. The model is additionally constrained to start and end in the background noise $s = 0$ state ($s_1 = s_N = 0$).

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

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:

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

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

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

In this section we first propose a new algorithm for solving the constrained Optimal Paritioning problem (for a particular penalty λ), and then we propose a new algorithm for solving the constrained Segment Neighborhood problem (for a given number P^* of desired peaks).

3.1. Generalized functional pruning optimal partitioning (GFPOP)

The proposed generalization of the FPOP algorithm (Maidstone *et al.* 2016) allows optimal inference in models with inequality constraints between adjacent means, such as problem (5). The resulting algorithm, which we call Generalized Functional Pruning Optimal Partitioning (GFPOP), can compute the maximum likelihood parameters for any model with constraints that can be represented as a state graph $G = (V, E)$ with vertices V for states and edges E for possible changepoints (e.g., up-down constrained model in Figure 1, right). The state graph G can be converted into a directed acyclic graph that represents the dynamic programming updates required to solve the problem, e.g., Figure 2 represents the computations required to solve the up-down constrained 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 updates, e.g., equations (12) and (13) for the up-down constrained model.

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

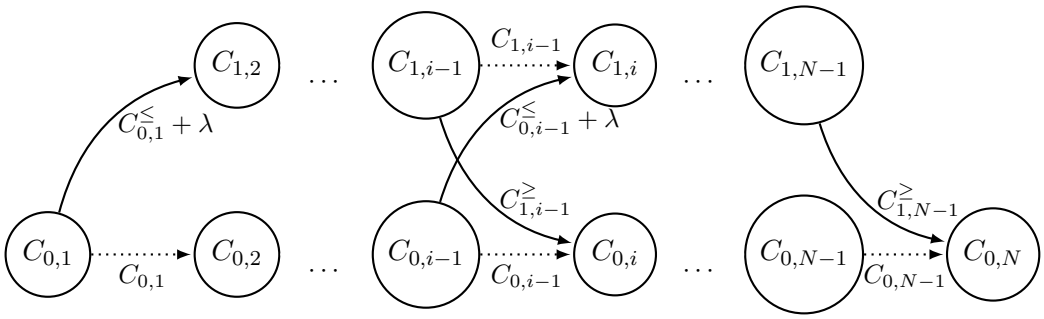


Figure 2: Directed acyclic graph (DAG) representing dynamic programming computations (Algorithm 1) for changepoint model with up-down constraints between adjacent segment means. Nodes in the graph repesent cost functions, and edges represent inputs to the the MinOfTwo sub-routine (solid=changepoint, 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 ($s_1 = 0$).

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.

Implementation details. As in PDPA (Rigaill 2015) and FPOP (Maidstone *et al.* 2016), the key idea is to use a data structure that provides an exact representation of the real-valued cost functions $C_{s,i}(\mu)$ (6). In the case of the Poisson loss ℓ, each $C_{s,i}(\mu)$ is a piecewise function, where each piece is of the form $\alpha\mu + \beta \log \mu + \gamma$, and $\alpha, \beta, \gamma \in \mathbb{R}$ are coefficients that exactly represent the cost function on a particular piece. Therefore we propose to store each optimal cost function $C_{s,i}(\mu)$ as an instance of a C++ `PiecewiseFunction` class, which is essentially a linked list of intervals of possible mean values μ, each with different coefficients α, β, γ. When the final cost function $C_{0,N}(\mu)$ has been computed, the solution to (5) is obtained by searching each piece in the linked list for the mean μ with min cost, $F(\lambda) = \min_\mu C_{0,N}(\mu)$. The implementation is thus highly dependent on the chosen loss function ℓ; implementing other convex loss functions (e.g., Gaussian) is possible but we save that for future work.

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: $C_{0,1} \leftarrow$ `PiecewiseFunction`$(z_1, \underline{z}, \overline{z})$
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: $\quad C_{1,i} \leftarrow \text{MinOfTwo}(M_1, C_{1,i-1}) +$ `PiecewiseFunction`$(z_i, \underline{z}, \overline{z})$
9: $\quad M_0 \leftarrow \text{MinMore}(i-1, C_{1,i-1})$ //cost of non-increasing change
10: $\quad C_{0,i} \leftarrow \text{MinOfTwo}(M_0, C_{0,i-1}) +$ `PiecewiseFunction`$(z_i, \underline{z}, \overline{z})$
11: mean, prevEnd, prevMean $\leftarrow \text{ArgMin}(C_{0,n})$ // begin decoding
12: seg $\leftarrow 1$; $U_{\text{seg}} \leftarrow$ mean; $T_{\text{seg}} \leftarrow$ 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: $\quad$ prevEnd, prevMean $\leftarrow \text{FindMean}(\text{mean}, \text{cost})$
17: $\quad$ seg $\leftarrow$ seg $+ 1$; $U_{\text{seg}} \leftarrow$ mean; $T_{\text{seg}} \leftarrow$ 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, which define the range of possible μ values (line 4). The optimal cost at the first data point is then initialized as a linked list with just one function piece, using the `PiecewiseFunction` constructor (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 $\mu \in (\underline{z}, \overline{z})$ and coefficients $(\alpha = 1, \beta = -z_1, \gamma = 0)$.

The dynamic programming recursion in this algorithm is a loop over data points i (line 6). Each iteration of the loop uses the MinLess/MinMore/MinOfTwo sub-routines to process the `PiecewiseFunction` objects. The details about how to implement these sub-routines have been described previously (Hocking *et al.* 2017). Briefly,

MinLess inputs a `PiecewiseFunction` $f(\mu)$ and outputs a `PiecewiseFunction` corresponding to the min-less operator $f^{\leq}(\mu)$ (11).

MinMore inputs a `PiecewiseFunction` $f(\mu)$ and outputs a `PiecewiseFunction` corresponding to the min-more operator $f^{\geq}(\mu)$ (14).

MinOfTwo inputs two `PiecewiseFunction` objects $f(\mu), g(\mu)$, and outputs a `PiecewiseFunction` corresponding to the pointwise minimum $\min\{f(\mu), g(\mu)\}$.

The DP update (12) is used to compute $C_{1,i}$: the penalty constant λ is added to the result of the MinLess sub-routine (line 7), before using the MinOfTwo sub-routine and adding the cost of the new data point (line 8). Similarly, the DP update (13) is used to compute $C_{0,i}$ on lines 9–10. It uses the MinMore sub-routine, does not add the penalty λ, then finally uses the MinOfTwo sub-routine.

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. Our disk-based implementation writes these parameters to a text output file.

Computational complexity. The time 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. Our implementation uses $O(NI)$ disk space and $O(I)$ memory. There are some pathological data sets for which the algorithm computes $I = O(N)$ intervals, which results in a worst case complexity of $O(N^2)$. One such example is when the penalty is large and the data sequence always increases, which also results in the worst case of the original FPOP algorithm. However our empirical study suggests much faster average complexity in real genomic data (Section 4.3).

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,

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

```
       chrom chromStart chromEnd count
      <char>      <int>    <int> <int>
   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, Koskas, Lebarbier, Rigaill, and Robin 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`:

```
R> data.dir <- file.path("Mono27ac", "chr11-60000-580000")
R> dir.create(data.dir, showWarnings = FALSE, recursive = TRUE)
R> PeakSegDisk::writeBedGraph(
+   Mono27ac$coverage, file.path(data.dir, "coverage.bedGraph"))
```

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

```
R> fit <- PeakSegDisk::PeakSegFPOP_dir(data.dir, 10000.5)
```

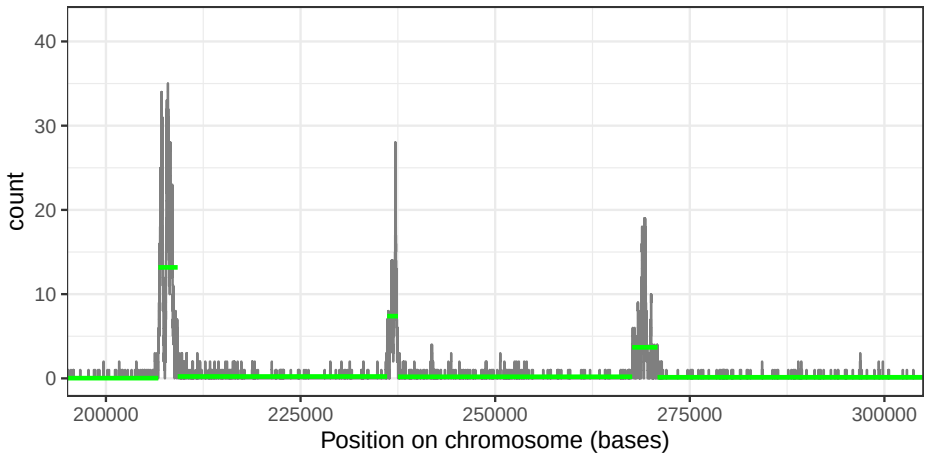
The first argument must be the folder name (not the `coverage.bedGraph` file name); the second argument must be a non-negative penalty λ value (larger penalties yield fewer peaks). The smallest value is `0` which yields max peaks, and the largest value is `Inf` which yields no peaks. For increased speed when solving for several penalties for the same problem, files are automatically created to store/cache the results. If the files already exist (and are consistent) then `PeakSegFPOP_dir` 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 `summary` method shown below returns one row of general information about the computed model:

```
R> summary(fit)

   penalty segments peaks  bases bedGraph.lines mean.pen.cost total.loss
     <num>    <int> <int>  <int>          <int>         <num>      <num>
1: 10000.5       15     7 520000           6921     0.2189399   43845.26
   equality.constraints mean.intervals max.intervals megabytes seconds
                  <int>          <num>         <int>     <num>   <num>
1:                    0        14.4279            41  11.03533   0.543
```

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

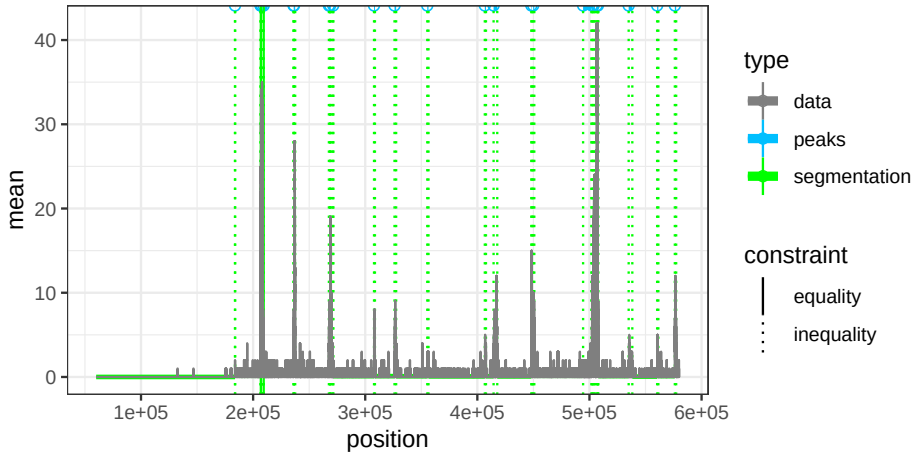
```
R> library("ggplot2")
R> gg <- ggplot() + theme_bw() + coord_cartesian(xlim = c(2e5, 3e5)) +
+   geom_step(aes(chromStart, count), color = "grey50",
+     data = Mono27ac$coverage) +
+   geom_segment(aes(chromStart, mean, xend = chromEnd, yend = mean),
+     color = "green", size = 1, data = fit$segments)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



It is apparent in the plot above that the peaks detected by our algorithm vary in both height (i.e., mean value) and length (in bases along the chromosome). Note that the typical "coverage" representation shown above results in a sequence of spatially correlated genomic data which are not independently distributed (many subsequent counts are identical). Although this lack of independence may be problematic for other statistical methods, our results indicate that our proposed algorithm yields similar peaks in data with or without such spatial correlation (Section 4.2).

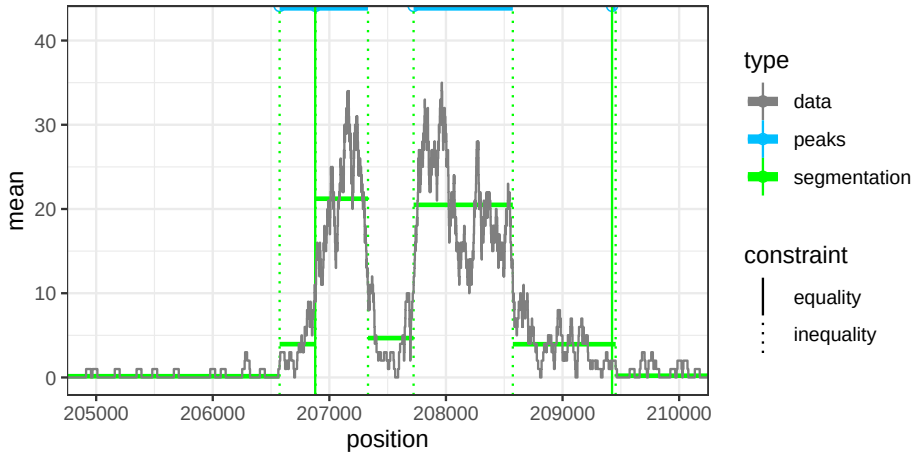
The **PeakSegDisk** package contains several other helper functions (which call the functions mentioned above). For example `PeakSegFPOP_vec` can handle integer vector data input (including automatic conversion of repeated values to a more efficient run-length encoding), and `PeakSegFPOP_df` takes a data frame as input:

```
R> fit <- PeakSegDisk::PeakSegFPOP_df(Mono27ac$coverage, 999.9)
R> class(fit)

[1] "PeakSegFPOP_df"  "PeakSegFPOP_dir" "list"

R> gg <- plot(fit)
R> print(gg)
```



Note that the special `plot` method above shows the data along with predicted peaks and segmentation model (which seems to have too many changepoints/peaks, suggesting that a larger penalty should be used). Vertical dashed green lines are used to show changepoint variables with strict inequality constraints, i.e., $c_i = 1, m_i < m_{i+1}$ or $c_i = -1, m_i > m_{i+1}$ in problem (5); vertical dashed solid lines show non-zero changepoint variables with equality constraints, i.e., $c_i \in \{-1, 1\}, m_i = m_{i+1}$. Since the `plot` method returns a `ggplot` object, it can be further processed, e.g., to zoom in on the part of the model with equality constraints:

```
R> print(gg + ggplot2::coord_cartesian(xlim = c(205000, 210000)))
```



The plot above shows that each equality constraint changepoint occurs just before another inequality constraint changepoint (first change up, second change down).

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),

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

From (17) it is 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 (18) 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

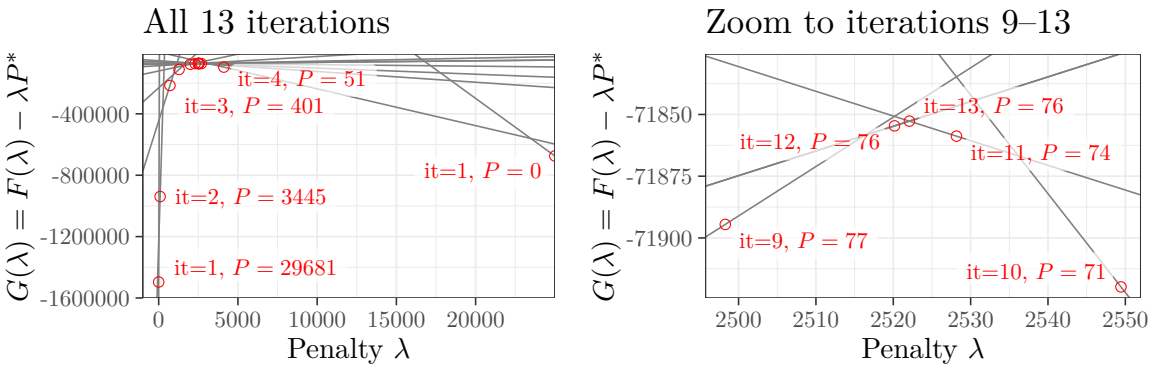


Figure 3: Example of a $G(\lambda)$ function which is maximized in order to find the most likely model with at most $P^* = 75$ peaks. Red dots show $G(\lambda)$ values evaluated by the algorithm; grey lines show affine functions $g_P(\lambda) = L_P + (P - P^*)\lambda$ used to determine the next λ value (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.

$\lambda_{\text{new}} = (L_{\overline{p}} - L_{\underline{p}})/(\underline{p} - \overline{p})$ (line 5). As previously described (Haynes, Eckley, and Fearnhead 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).

Computational complexity. The space complexity is the same as GFPOP: $O(NI)$ disk and $O(I)$ memory for N data and I intervals (candidate changepoints). Its time complexity is $O(WNI)$ where W is the number of times the GFPOP sub-routine is called, i.e., the number of iterations of the while loop (line 4). Because the algorithm essentially performs binary search, we expect $W = O(\log P_{\max})$ iterations, where $P_{\max} = O(N)$ is the maximum number of peaks for the data set. We study the empirical number of iterations W in Sections 4.5–4.6.

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

```
R> fit <- PeakSegDisk::sequentialSearch_dir(data.dir, 17L)
```

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

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

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:

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

```
   iteration under  over   penalty peaks  total.loss
       <num> <int> <int>     <num> <int>       <num>
1:         1    NA    NA    0.0000  3199 -130227.291
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 $N = 10^7$ lines in the bedGraph file. Note that our implementation of the proposed algorithm uses the previously described cost function weighting/compression technique (Cleynen *et al.* 2014; Hocking *et al.* 2017), so the number of dynamic programming updates N is equal to the number of lines in the bedGraph file (which is much smaller than the number of bases). For example if the data sequence 5,1,1,1,0,0,5,5 is encoded as $N = 4$ lines z_i 5,1,0,5 with corresponding weights w_i 1,3,2,2 then the weighted/compressed Poisson loss function for mean μ is $\ell(z_i, w_i, \mu) = w_i(\mu - z_i \log \mu)$. Therefore in the following sections our analysis of computational complexity refers to data set size N in terms of number of lines in the bedGraph file.

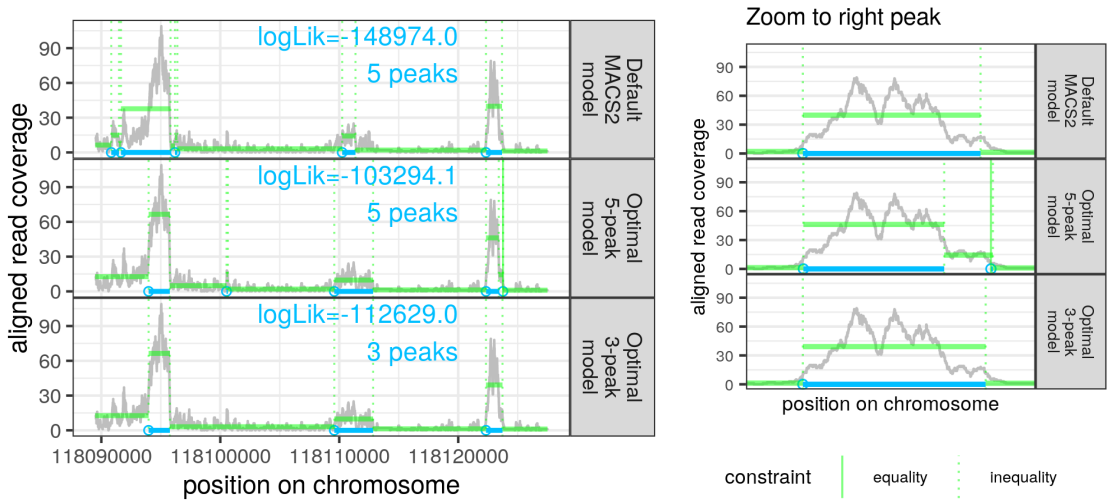


Figure 4: One ChIP-seq data set with three peak models. (green horizontal segment means; green dotted vertical lines for changepoints; blue bars for peaks; blue dots for peak starts) **Top:** the MACS2 algorithm (a heuristic from the bioinformatics literature) computed a sub-optimal model with five peaks 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.

4.1. Computing maximum likelihood models 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 bioinformatics literature (Zhang, Liu, Meyer, Eeckhoute, Johnson, Bernstein, Nusbaum, Myers, Brown, Li *et al.* 2008). Using default parameter settings MACS2 detects five peaks, so we ran our proposed sequential search (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 active 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. The given example shows that our algorithm 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 robust to spatial correlation in typical genomic data

As in all genomic data which use the typical "coverage" representation of DNA sequence reads aligned to a reference genome, there is spatial correlation in the ChIP-seq data we analyze. In other words, a sequence read is counted at *each genomic position* where it aligns to the reference genome, which results in a sequence of counts which is not independent. For example

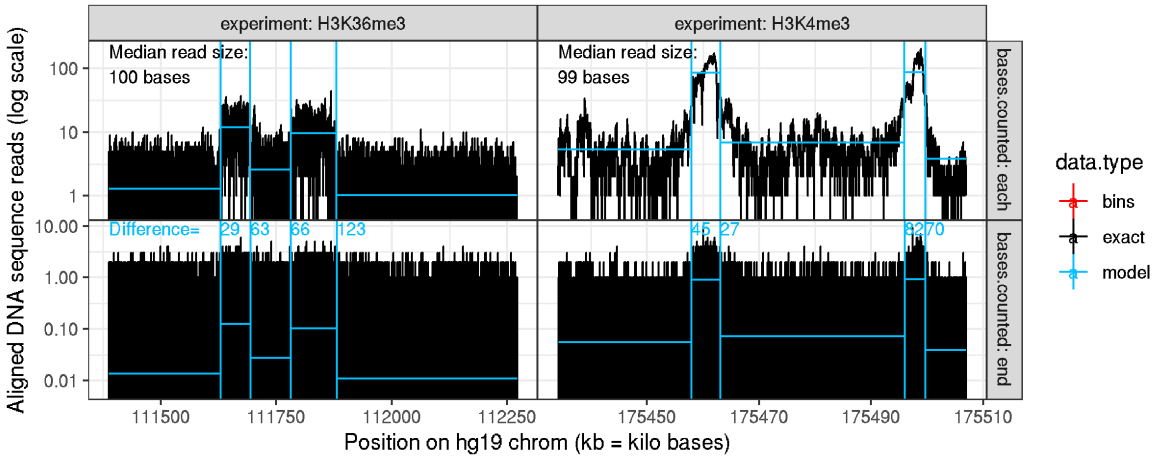


Figure 5: For two different ChIP-seq experiment types (left=broad H3K36; right=sharp H3K4), our proposed GFPOP algorithm detects similar peaks (blue) in data sequences with (top) or without (bottom) spatial correlation. Also shown are raw/exact count data (black), mean counts in bins (red), and peak position differences in bases between top/bottom panels (blue numbers).

if a read of size 100 bases aligns on chr1:1-100 and another read aligns on chr1:51-150, then the typical coverage representation is a count of 1 aligned read on chr1:1-50, a count of 2 aligned reads on chr1:51-100, and a count of 1 aligned read on chr1:101-150, which leads to spatial correlation between nearby positions (top panels of Figure 5). Another representation with no spatial correlation would be to count a sequence read at *only the last genomic position* where it aligns (bottom panels of Figure 5). In the example above that would be counting 1 at chr1:100 and 1 at chr1:150, and 0 elsewhere.

We have tried GFPOP on both representations of the data, and we have observed that the detected peaks are highly consistent (with some variation on the order of the read size, 100 bases). In Figure 5 we represent two datasets (left:H3K36 and right:H3K4) using the coverage representation (top) and last position representation (bottom). The peak positions of the models shown in blue are highly consistent between the top and bottom plots. This illustrates that the proposed algorithm is highly robust to spatial correlation.

4.3. 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 6, 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

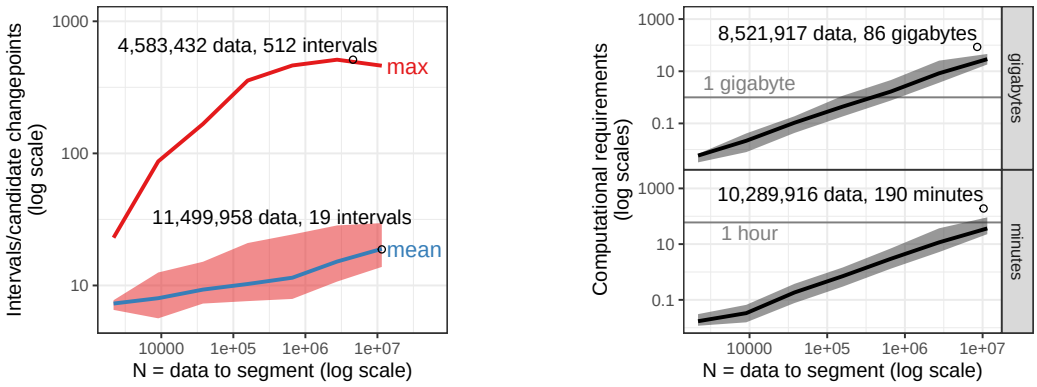


Figure 6: In our empirical tests, the computational requirements of the GFPOP algorithm were log-linear $O(N \log N)$ in the number of data points N to segment. **Left:** we analyzed the number of intervals I (candidate changepoints) stored in the $C_t(\mu)$ cost functions, because the total time/space 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 and penalties λ; black dots and text show computational requirements for the most extreme data sets.

consistent with this expectation (Figure 6, 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.

4.4. 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 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. Note that these results depend on the disk type, the amount of RAM available for disk caching, and the access pattern; the ATA disk we used was a TOSHIBA MK1652GSX (attached to SCSI, cache/buffer size = 8192 KB). We observed that disk storage is slower than memory storage by a constant factor (1.7–2.3×, Figure 7), which was expected.

4.5. Sequential search is faster than Segment Neighborhood

In this section we compare the number of GFPOP calls 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 empirically an $O(N \log N)$ operation. In contrast, the proposed sequential search (Algorithm 2) needs to solve for a sequence of

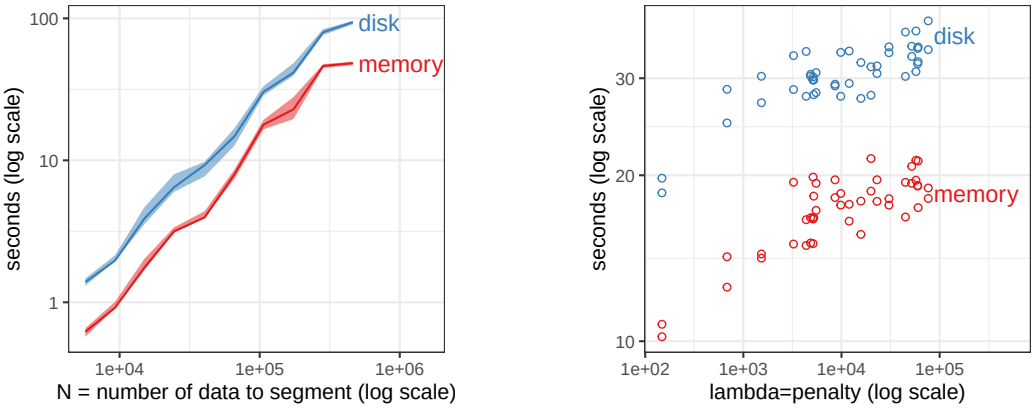


Figure 7: The disk-based storage method is only a constant factor slower than the memory-based method. We benchmarked both methods on several small data sets ($N \leq 462,890$) for which optimal models could be computed using 1GB of storage. **Left:** computation time is empirically $O(N \log N)$ for 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 with $N = 106,569$, the computation time generally increases with penalty value λ for both storage methods.

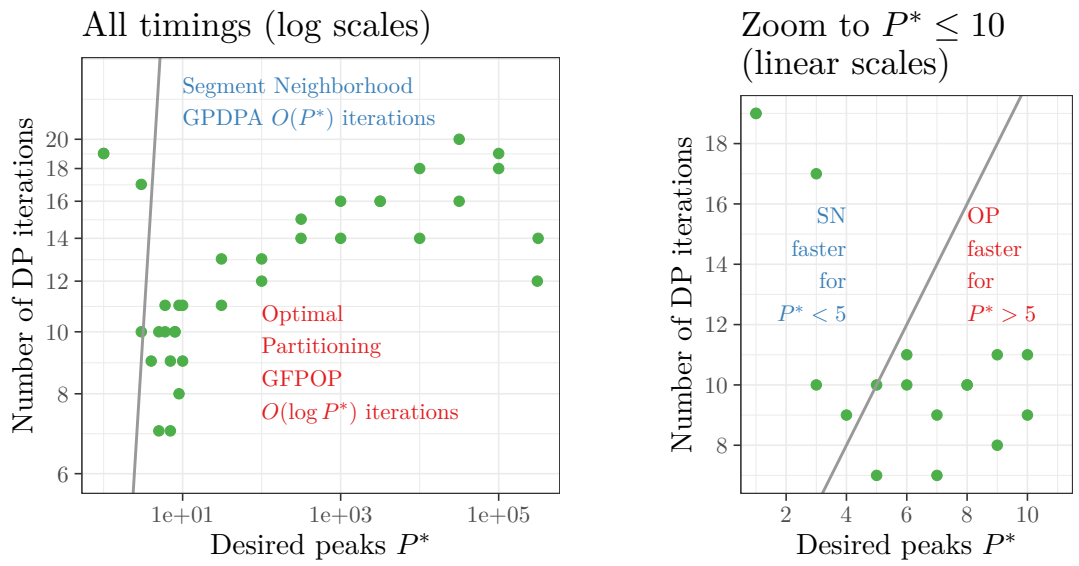


Figure 8: Comparison of time to compute optimal model with at most P^* desired peaks using Segment Neighborhood (grey lines) and Optimal Partitioning with proposed sequential search (green dots). GFPOP with sequential search (Algorithm 2) was used to compute optimal models with different numbers of desired 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 (11 segments); Segment Neighborhood is faster for smaller models.

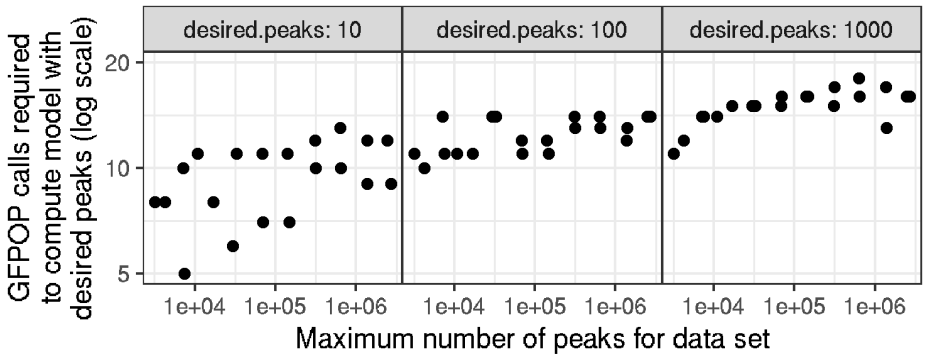


Figure 9: The number of GFPOP calls required during the sequential search algorithm is $O(\log P_{\max})$ where $P_{\max}$ is the maximum number of peaks for a data set. Each panel shows, for a fixed input parameter P^* (desired number of peaks), the empirical number of GFPOP calls over several data sets of different sizes.

penalties, each of which is done via GFPOP in empirically $O(N \log N)$ 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 8, left). For a large desired number of peaks ($P^* > 5$), it is clearly faster to use the sequential search algorithm (Figure 8, right). Overall these experiments indicate that the sequential search for P^* desired peaks takes $O(\log P^*)$ GFPOP calls.

4.6. Sequential search iterations increase logarithmically with data set size

Recall that the proposed sequential search (Algorithm 2) repeatedly calls the GFPOP subroutine until it finds the desired model with P^* peaks. Because the algorithm is an instance of binary search, we expected the number of GFPOP calls W to increase logarithmically with the maximum number of peaks $P_{\max}$. To test this hypothesis we ran the sequential search algorithm on several ChIP-seq data sets from the UCI benchmark, with $P_{\max}$ varying from about $10^{3.5}$ to about $10^{6.5}$. For each data set we determined the maximum number of peaks $P_{\max}$ by running GFPOP with a penalty of $\lambda = 0$. We then ran the sequential search algorithm on each data set with $P^* \in \{10, 100, 1000\}$ as values for the input parameter (desired number of peaks). We observed in these data that the number of GFPOP calls increases logarithmically with the maximum number of peaks (Figure 9). These results provide a substantial empirical validation that the sequential search takes $O(\log P_{\max})$ GFPOP calls, as expected.

4.7. 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. These labels were created by expert genomic scientists, who used visual inspection of coverage data plots to determine genomic regions with/without peaks which are significantly higher than the neighboring background noise. Although such labels are not available in all ChIP-seq data sets, here we use them as a gold standard to compute false positive and false

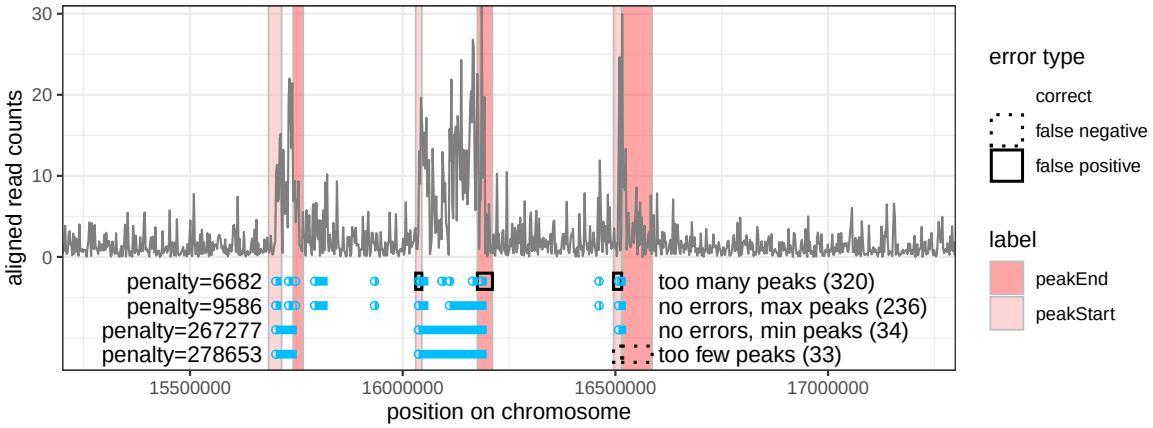


Figure 10: Labels are used to compute an error rate for each peak model (blue bars), defined as the sum of false positive and false negative labels (rectangles with black outline). This H3K36me3 ChIP-seq data set has $N = 1,254,751$ data to segment on a subset of chr12, but in the plot above we show 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.

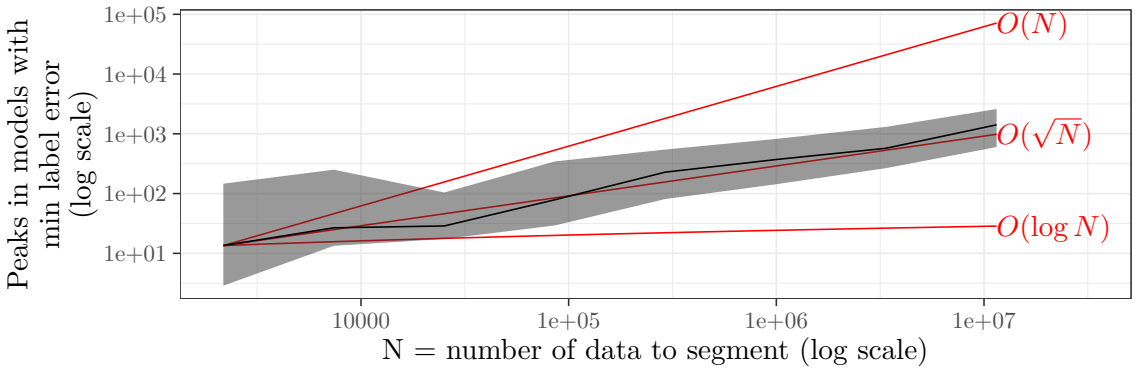


Figure 11: The model with minimal label errors has $O(\sqrt{N})$ peaks in a data set of size N. For each data set we computed peak models with minimal label errors (see Figure 10); we then plot the number of peaks in minimal error models as a function of data set size N. Black median line and grey quartile band computed over several data sets of a given size N; asymptotic reference lines shown in red.

negative rates for peak models. For example Figure 10 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 desired number of peaks P^* (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

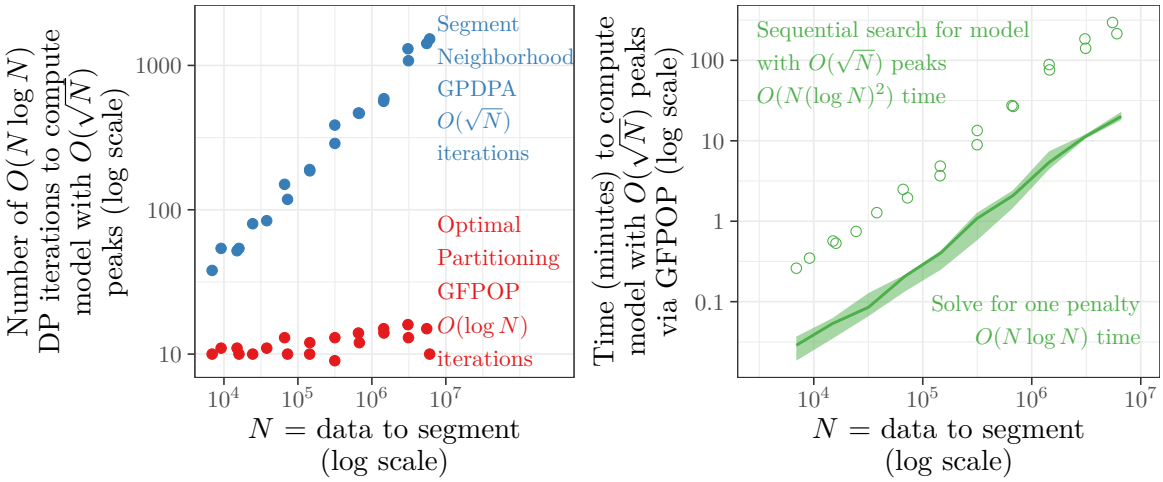


Figure 12: Computing a zero-error model with $O(\sqrt{N})$ peaks is possible in $O(N(\log N)^2)$ time using our proposed Optimal Partitioning Search algorithm. **Left:** Segment Neighborhood requires $O(\sqrt{N})$ dynamic programming iterations to compute a model with $O(\sqrt{N})$ peaks; our proposed 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).

that achieves zero label errors (34/236 in Figure 10), 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 N in Figure 11. From the figure it is clear that the mean number of peaks required to achieve zero label errors in these data is on the order of $O(\sqrt{N})$.

Computing the optimal model with $P = O(\sqrt{N})$ 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, $P^* = 1414$ peaks are desired in order to achieve 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 6). Overall that would mean 220 terabytes of storage and 17 weeks of computation time, which is much too expensive in practice.

Instead, we propose to use the sequential search (Algorithm 2) to compute the model with the desired number of peaks. In our empirical tests, we observed that only $O(\log N)$ GFPOP calls are required to compute $O(\sqrt{N})$ peaks (Figure 12, 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 12, 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 Pruning Optimal Partitioning (GFPOP) which computes the optimal model for one penalty λ. We also proposed a sequential search algorithm which repeatedly calls GFPOP with different penalties λ, in order to compute the most likely model with at most P^* desired peaks ($K = 2P^* + 1$ segments). The algorithms are implemented in the R package **PeakSegDisk** which is publicly available on CRAN and GitHub.

This paper has several substantial similarities and differences with previous work. The main new algorithm presented in this paper is GFPOP, which generalizes the FPOP algorithm of Maidstone *et al.* (2016). FPOP was originally proposed for use with Gaussian mean change-point problems, but has also been used with robust loss functions, resulting in the R-FPOP algorithm (Fearnhead and Rigaill 2019). FPOP and R-FPOP perform the same dynamic programming updates, but use different representations of the cost function. In contrast, in this paper we propose a generalization of FPOP for changepoint problems with inequality constraints between adjacent segment means. The resulting algorithm, GFPOP, is substantially novel because it uses different dynamic programming update rules, which depend on the constraints. We refer to the algorithm as "Generalized" because FPOP is the special case corresponding to a constraint graph with only one node and edge (Figure 1, left). The up-down constrained model we implemented in **PeakSegDisk** is another special case, but the GFPOP algorithm works for any model which can be represented as a state graph (Figure 1).

Whereas we previously proposed an exact solver for the up-down constrained changepoint model (Hocking *et al.* 2017), the current paper provides a more efficient algorithm for computing the same model on the large genomic data sets that are now common. In such data, 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, $O(N \log N)$ disk, and $O(\log N)$ memory, which makes it possible to compute optimal models with many peaks on common laptop computers. More precisely, the novelty of this paper with respect to that previous work is in terms of:

Data. Both papers propose peak detection algorithms for genomic data. Our previous work analyzes a smaller data set (2752 sequences of sizes up to $N = 10^5$) whereas in this paper we analyze a larger data set (4951 sequences of sizes up to $N = 10^7$).

Problem formulation. Both papers analyze changepoint problems with up-down constraints between adjacent segment means. Our previous work focuses on solving the Segment Neighborhood problem (best model in K segments), whereas in this paper we additionally solve the Optimal Partitioning problem (best model for penalty λ).

Algorithm complexity. Both papers propose dynamic programming algorithms for computing the same up-down constrained changepoint model. The algorithm in our previous work is $O(KN \log N)$ which is too slow when both the number of segments K and data N are large. This paper proposes an $O(N \log N)$ algorithm which is useful for computing complex models of large data sets.

Results/baselines. The empirical analysis in our previous paper demonstrates more accurate peak detection in cross-validation experiments; we considered baseline algorithms from the bioinformatics literature, and baseline changepoint algorithms (unconstrained, heuristic). The empirical analysis in this paper demonstrates faster computation time in large data, relative to the up-down constrained Segment Neighborhood algorithm.

Finally the previous work of Haynes *et al.* (2017) has proposed the CROPS algorithm for computing all changepoint models for a range of penalties $(\underline{\lambda}, \overline{\lambda})$ (which is the input parameter). In this paper we use similar ideas in our proposed sequential search algorithm for computing the optimal model with at most P^* desired peaks (which is the input parameter). Both algorithms work by repeatedly solving the Optimal Partitioning problem for different penalties λ.

In an analysis of benchmark genomic data sets, we empirically showed that our sequential search algorithm only requires $O(\log P^*)$ calls to GFPOP. For example, this approach can compute an optimal model with ≈ 1000 peaks for $N = 10^7$ data using only ≈ 20 calls to GFPOP. This translates into only hours of compute time and gigabytes of storage, which is much less than weeks/terabytes which would be required for the previous Segment Neighborhood solver. More generally, these results suggest that our proposed sequential search algorithm can be used to reduce computation times for other changepoint models, i.e., Optimal Partitioning combined with the sequential search algorithm should be preferred over the classical Segment Neighborhood approach.

For future work we look forward to implementing our proposed disk-based storage mechanism for other loss/likelihood functions and constraints. For example, we are currently working on a **gfpop** package which uses the Gaussian loss/likelihood with a user-specified constraint graph. Results from this paper indicate that disk-based storage can be used to scale to very large data sets, while maintaining fast computation times.

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

Acknowledgements. 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.

References

Arlot S, Brault V, Baudry JP, Maugis C, Michel B (2016). ***capushe**: CAlibrating Penalities Using Slope HEuristics*. R package version 1.1.1, URL https://CRAN.R-project.org/package=capushe.

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

Baranowski R, Fryzlewicz P (2019). ***wbs**: Wild Binary Segmentation for Multiple Change-Point Detection*. R package version 1.4, URL https://CRAN.R-project.org/package=wbs.

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

Bellman R (1961). "On the Approximation of Curves by Line Segments Using Dynamic Programming." *Commun. ACM*, **4**(6), 284–.

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

Choi H, Nesvizhskii AI, Ghosh D, Qin ZS (2009). "Hierarchical hidden Markov model with application to joint analysis of ChIP-chip and ChIP-seq data." *Bioinformatics*, **25**(14), 1715–1721.

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

Fearnhead P, Rigaill G (2019). "Changepoint Detection in the Presence of Outliers." *Journal of the American Statistical Association*, **114**(525), 169–183.

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

Fryzlewicz P (2014). "Wild Binary Segmentation for multiple change-point detection." **42**.

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

Hocking T, Rigaill G, Fearnhead P, Bourque G (2017). "A log-linear time algorithm for constrained changepoint detection." ArXiv:1703.03352.

Hocking TD (2018). ***PeakSegOptimal**: Optimal Segmentation Subject to Up-Down Constraints*. R package version 2018.05.25.

Hocking TD (2019). ***binseg**: Binary Segmentation*. R package version 2019.9.4, URL https://github.com/tdhock/binseg.

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

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

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

Killick R, Eckley IA (2014). "**changepoint**: An R Package for Changepoint Analysis." *Journal of Statistical Software*, **58**(3), 1–19. URL http://www.jstatsoft.org/v58/i03/.

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

Lebarbier É (2005). "Detecting multiple change-points in the mean of Gaussian process by model selection." *Signal processing*, **85**(4), 717–736.

Li H, Munk A, Sieling H (2016). "FDR-control in multiscale change-point segmentation." *Electron. J. Statist.*

Li H, Sieling H (2017). ***FDRSeg**: FDR-Control in Multiscale Change-Point Segmentation*. R package version 1.0-3, URL https://CRAN.R-project.org/package=FDRSeg.

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

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

Newman CBD, Merz C (1998). "UCI Repository of machine learning databases."

Pein F, Hotz T, Sieling H, Aspelmeier T (2019). ***stepR**: Multiscale change-point inference*. R package version 2.0-4, URL https://CRAN.R-project.org/package=stepR.

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

Rigaill G (2015). "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).

Rigaill G, Hocking T, Vert JP, Bach F (2013). "Learning sparse penalties for change-point detection using max margin interval regression." In *Proc. 30th ICML*, pp. 172–180.

Schwarz G (1978). "Estimating the Dimension of a Model." *Ann. Statist.*, **6**(2), 461–464.

Wilibanks E, Facciotti M (2010). "Evaluation of Algorithm Performance in ChIP-Seq Peak Detection." *PLoS ONE*, **5**(7).

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

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

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

Affiliation:

Toby Dylan Hocking
Northern Arizona University
School of Informatics, Computing, and Cyber Systems
Flagstaff, AZ, USA
E-mail: Toby.Hocking@nau.edu

Guillem Rigaill
Laboratoire de Mathématiques et Modélisation d'Evry (LaMME)
Université d'Evry Val d'Essonne, INRAE
Evry, France
or
Institute of Plant Sciences Paris Saclay IPS2
CNRS, INRAE, Université Paris-Sud, Université Evry, Université Paris-Saclay
Batiment 630, 91405 Orsay, France.
or
Institute of Plant Sciences Paris-Saclay IPS2
Paris Diderot, Sorbonne Paris-Cité
Bâtiment 630, 91405, Orsay, France.
E-mail: Guillem.Rigaill@inra.fr

Paul Fearnhead
Lancaster University
Lancaster, UK
E-mail: p.fearnhead@lancaster.ac.uk

Guillaume Bourque
McGill University
Montréal, Québec, Canada
E-mail: guil.bourque@mcgill.ca

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd