Taylor & Francis
Taylor & Francis Group

Journal of Computational and Graphical Statistics

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/ucgs20

Linear Time Dynamic Programming for Computing Breakpoints in the Regularization Path of Models Selected From a Finite Set

Joseph Vargovich & Toby Dylan Hocking

To cite this article: Joseph Vargovich & Toby Dylan Hocking (2022) Linear Time Dynamic Programming for Computing Breakpoints in the Regularization Path of Models Selected From a Finite Set, Journal of Computational and Graphical Statistics, 31:2, 313-323, DOI: 10.1080/10618600.2021.2000422

To link to this article: https://doi.org/10.1080/10618600.2021.2000422

JOURNAL OF COMPUTATIONAL AND GRAPHICAL STATISTICS
2022, VOL. 31, NO. 2, 313–323
https://doi.org/10.1080/10618600.2021.2000422

Linear Time Dynamic Programming for Computing Breakpoints in the Regularization Path of Models Selected From a Finite Set

Joseph Vargovich and Toby Dylan Hocking

SICCS Machine Learning Research Laboratory, Northern Arizona University, Flagstaff, AZ

ABSTRACT

Many learning algorithms are formulated in terms of finding model parameters which minimize a data-fitting loss function plus a regularizer. When the regularizer involves the ℓ_0 pseudo-norm, the resulting regularization path consists of a finite set of models. The fastest existing algorithm for computing the breakpoints in the regularization path is quadratic in the number of models, so it scales poorly to high-dimensional problems. We provide new formal proofs that a dynamic programming algorithm can be used to compute the breakpoints in linear time. Our empirical results include analysis of the proposed algorithm in the context of various learning problems (regression, changepoint detection, clustering, and matrix factorization). We use a detailed analysis of changepoint detection problems to demonstrate the improved accuracy and speed of our approach relative to grid search and a previous quadratic time algorithm.

ARTICLE HISTORY
Received July 2020
Revised August 2021

KEYWORDS
Binary segmentation;
Changepoint detection;
Dynamic programming;
Model selection

1. Introduction

In statistical machine learning, two central concepts are optimization and regularization. Optimization is typically used to compute model parameters which result in the best predictions on a training set, whereas regularization is used to avoid overfitting, which occurs when predictions are only accurate for training data and not for held-out validation or test data. There are many types of regularization, but the focus in this article is on methods which measure model complexity by counting the number of nonzero parameters. Some typical examples are given in Table 1, which includes best subset regression (Miller 2002; Soussen et al. 2010), optimal segmentation (Auger and Lawrence 1989; Jackson et al. 2005), k-means clustering (MacQueen 1967), and low-rank matrix factorization (Huang and Wolkowicz 2018). In this context, the regularization path is a sequence of models with varying complexity, for example, from $k = 1$ to 20 cluster centers in k-means clustering. Another example of the regularization path is the sequence of optimal segmentation models selected for a given range of penalties (Haynes, Eckley, and Fearnhead 2017).

More precisely, we propose to study the regularization path of machine learning problems such as

$$\hat{\theta}(\lambda) = \arg\min_\theta \mathcal{L}(\theta) + \lambda R(\theta), \tag{1}$$

where $\theta \in \mathbb{R}^p$ is a vector of model parameters, the loss function $\mathcal{L} : \mathbb{R}^p \to \mathbb{R}$ is typically convex, and $\lambda \geq 0$ is a penalty constant. The regularizer $R : \mathbb{R}^p \to \mathbb{R}_+$ is a non-convex function involving the ℓ_0 pseudo-norm,

$$||\theta||_0 = \sum_{j=1}^{p} I[\theta_j \neq 0] \in \mathbb{Z}_+ = \{0, 1, \ldots, p\} \tag{2}$$

which counts the number of nonzero entries of the θ parameter vector (I is the indicator function). For some learning problems, it is important to compute not just a single model for one penalty λ, but also the full regularization path $\{\hat{\theta}(\lambda)|\lambda \geq 0\} = \{\Theta_1, \Theta_2, \ldots, \Theta_N\}$. The path is a finite set of N models, that is, for any $\lambda \geq 0$, we have $\hat{\theta}(\lambda) = \Theta_k$ for some model size $k \in \{1, \ldots, N\}$. To simplify the presentation we limit our discussion to regularizers $R(\Theta_k) = k$ which are equal to model size. More general regularizers, for example, $R(\Theta_k) = r_k$ for some sequence of increasing values $r_1 < \cdots < r_N$, can be handled using a straightforward modification of our proposed algorithm, which is described in Section 2.2.

One example is the best subset regression, which seeks the best k features for a linear regression function. Typically the loss function is mean-squared prediction error, and the regularizer is the number of non-zero entries (active features) in the learned weight vector. In this problem, there a total of p input features and therefore a set of $N = p + 1$ model sizes in the regularization path $\{\Theta_0, \Theta_1, \ldots, \Theta_p\}$. This nonconvex problem is NP-hard, so the optimal regularization path can only be computed for low-dimensional problems (Bertsimas, King, and Mazumder 2016). For high-dimensional problems there are various heuristic algorithms for computing approximate solutions, for example, greedy forward/backward selection (Mallat and Zhang 1993; Davis, Mallat, and Zhang 1994; Miller 2002; Schniter, Potter, and Ziniel 2009; Soussen et al. 2010), non-smooth non-convex regularizers (Fan and Li 2001; Zhang 2010; Mazumder, Friedman, and Hastie 2011; van den Burg, Groenen, and Alfons 2017), and ℓ_1 regularization/LASSO (Tibshirani 1996; Chen, Donoho, and Saunders 1998). Each weight vector $\Theta_k \in \mathbb{R}^p$ in the optimal or approximate regularization path has

CONTACT Joseph Vargovich jrv233@nau.edu School of Informatics, Computing, and Cyber Systems, Northern Arizona University, 1295 Knoles Dr, Flagstaff, AZ 86011.

Table 1. Examples of machine learning problems which use the ℓ_0 pseudo-norm as a regularizer.

Problem	Loss	Regularizer	Model complexity
Best subset regression	$\|\|X\theta - y\|\|_2^2$	$\|\|\theta\|\|_0$	Features selected
Optimal segmentation	$\|\|\theta - y\|\|_2^2$	$\|\|D\theta\|\|_0$	Segments/changepoints
k-means Clustering	$\|\|\theta M - X\|\|_F$	$\|\|\theta^T \mathbf{1}\|\|_0$	Cluster centers
Matrix factorization	$\|\|U\text{Diag}(\theta)V^T - X\|\|_F$	$\|\|\theta\|\|_0$	Rank

k nonzero entries, $R(\Theta_k) = ||\Theta_k||_0 = k$. The extreme elements correspond to the ordinary least squares solution Θ_p with all features selected, and the completely regularized solution Θ_0 with no features selected.

Another example is optimal segmentation, which is the maximum likelihood model with k segments ($k-1$ changepoints) for a sequential dataset. In this problem, there are p sequence data, and each element Θ_k of the regularization path has $k \in \{1, \ldots, p\}$ distinct segments ($k-1$ changes) along the sequence, that is, $R(\Theta_k) = 1 + ||D\Theta_k||_0 = k$ where $D \in \mathbb{R}^{p-1 \times p}$ is the matrix which returns the difference between adjacent pairs of data in the sequence. Even though this problem is nonconvex, an optimal solution can be computed via dynamic programming algorithms that are log-linear in the number of data, and linear in the number of models (Killick, Fearnhead, and Eckley 2012; Maidstone et al. 2016). There are also several fast heuristic algorithms, including binary segmentation (Scott and Knott 1974; Truong, Oudre, and Vayatis 2018), which computes an approximate regularization path of $N = p$ models in $O(N \log N)$ time on average and $O(N^2)$ in the worst case. The optimal or approximate regularization path of $N = p$ models $\{\Theta_1, \Theta_2, \ldots, \Theta_p\}$ has extreme elements Θ_1 with no changes (one common segment/parameter for the entire data sequence) and Θ_p with a change after every data point (a different segment/parameter for each data point).

In this context, solving the penalized problem (1) for a given penalty $\lambda \geq 0$, results in one of the solutions to the corresponding constrained problem,

$$L_k = \min_{\theta} \mathcal{L}(\theta), \text{ subject to } R(\theta) \leq k, \tag{3}$$

where $k \in \mathbb{Z}_+$ is the model size (selected features, changepoints, clusters, etc). We assume there is some algorithm that can compute a regularization path $\{\Theta_1, \Theta_2, \ldots, \Theta_N\}$ of solutions to (3) with corresponding loss values, $L_1 > \cdots > L_N$ (e.g., binary segmentation algorithm for changepoint detection, k-means algorithm for clustering).

In this article, we use the term "model selection" to refer to an algorithm for computing a desirable model size $k \in \mathbb{Z}_+$, given precomputed loss values $L_1, \ldots, L_N$. If the loss values $L_1 > \ldots > L_N$ in a regularization path $\{\Theta_1, \Theta_2, \ldots, \Theta_N\}$ are known, they can be used to define the model selection function

$$k_N^*(\lambda) = \arg\min_{k \in \{1,\ldots,N\}} \underbrace{L_k + \lambda k}_{f_k(\lambda)}. \tag{4}$$

The model selection function (4) returns the (smallest) model complexity k which is optimal for a given penalty parameter λ. Model complexity refers to the number of segments used to fit blocks of data, separated by detected changepoints, with an average value line. Thus, the model complexity (k) refers to the total amount of unique segments present in a given segmentation model. Note that strictly speaking the model selection function selects the order or size k of the model (not the model parameters Θ_k, which are assumed to be computed prior to selection). In this article, we provide a new formal proof that dynamic programming can be used to compute an exact representation of the model selection function $k_N^*(\lambda)$ in linear $O(N)$ time, as well as the set of corresponding breakpoints between models selected (assuming that the loss values L_k have already been computed).

1.1. Existing Algorithms and Related Work

The model selection function $k_N^*(\lambda)$ can be trivially evaluated for a single λ parameter in linear $O(N)$ time, which yields the solution to (1) via $\hat{\theta}(\lambda) = \Theta_{k_N^*(\lambda)}$. However, for some learning algorithms, we need an exact representation of the model selection function for a full path of penalty λ values. Given a set of N precomputed loss values L_k, there are quadratic $O(N^2)$ time algorithms for computing the model selection function, which were proposed in the context of changepoint detection (Lavielle 2005; Hocking et al. 2013) and regression (Arlot and Massart 2009). These quadratic time algorithms are not based on dynamic programming (see Section 3.2 for details), and are too slow for high-dimensional problems such as computing the full path of binary segmentation models for large datasets. In this context computing, the N loss values can be done in log-linear $O(N \log N)$ time, so the quadratic time model selection is the speed bottleneck (see Section 4.2).

The algorithm we propose is similar to the "convex hull trick" which is informally described, without any references to the machine learning literature, on a web page (PEGWiki 2018). The novelty of our article with respect to that previous work is (i) rigorous formal proofs of the linear time complexity and optimality, (ii) explaining the relevance to the machine learning literature, and (iii) detailed theoretical and empirical comparisons with baseline algorithms.

A final related work is the CROPS algorithm of Haynes, Eckley, and Fearnhead (2017), which also proposes an algorithm that outputs an exact path of solutions for several penalty values. Both articles exploit the structure of the piecewise linear model selection function which relates the constrained and penalized problems. The input to our algorithm is a sequence of constrained models of sizes 1 to N, whereas the input to CROPS is an interval of penalty values. In general the two algorithms output different results (partial solution paths). However, in the special case, when $N = p$ models are input to our algorithm (all possible models) and the interval $(0, \infty)$ is input to CROPS, then the two algorithms return the same output (the full path).

1.2. Contributions and Organization

In this article, we propose a new dynamic programming algorithm for computing the model selection function $k_N^*(\lambda)$, and we prove that it computes an exact representation for all penalties $\lambda \geq 0$ (Section 2). Our second contribution is a theoretical analysis of the time complexity of our algorithm, which demonstrates that it is linear $O(N)$ time in the worst case; we also provide a theoretical analysis of previous algorithms in terms

of the framework of this article (Section 3). Our third contribution is an empirical study of time complexity in several real and synthetic datasets, including a comparison with previous algorithms (Section 4). Our final contribution is an empirical study of the prediction accuracy in cross-validation experiments on supervised changepoint detection problems (Section 5.1). The article concludes with a discussion (Section 6).

2. Proposed Dynamic Programming Algorithm

We propose a dynamic programming algorithm for N decreasing loss values $L_1 > \cdots > L_N$; it computes an exact representation of the $k^*_N(\lambda)$ model selection function for all penalties $\lambda \geq 0$.

2.1. Exact Representation of a Piecewise Constant Function Using Breakpoints

Our proposed algorithm recursively computes k^*_N from k^*_{N-1}, so is an instance of dynamic programming (Bellman 1961). At each step/iteration $t \in \{1, \ldots, N\}$ of the algorithm, the algorithm stores a set of $M_t \in \{1, \ldots, t\}$ selectable models,

$$1 = K_{t,1} < K_{t,2} < \cdots < K_{t,M_t} = t. \tag{5}$$

Note that when all model sizes from 1 to t are selected for at least one penalty λ then we have $M_t = t$. Otherwise, if at least one model is not selected for any penalty λ, then we have fewer selectable models, $M_t < t$. The algorithm also stores a corresponding set of breakpoints,

$$\infty = b_{t,0} > b_{t,1} > \cdots > b_{t,M_t} = 0. \tag{6}$$

These two sets define for all $t \geq 1$ a recursively computed model selection function,

$$F_t(\lambda) = \begin{cases} K_{t,1} & \text{if } \lambda \in (b_{t,1}, b_{t,0}) \\ \quad\vdots & \\ K_{t,M_t} & \text{if } \lambda \in (b_{t,M_t}, b_{t,M_t-1}) \end{cases} \tag{7}$$

We prove later (Theorem 1) that the recursively computed function F_N is identical to k^*_N, the desired model selection function (4). The geometric interpretation of the models $K_{t,i} \in \mathbb{Z}$ and breakpoints $b_{t,i} \in \mathbb{R}$ are shown in Figure 1. Each breakpoint is a penalty value where the min cost (gray segments) changes from one cost function to another (black lines).

2.2. Dynamic Programming Update Rules

The algorithm starts at the first iteration $t = 1$ by initializing $M_1 = 1$ model with

$$K_{1,1} = 1,\ b_{1,1} = 0,\ b_{1,0} = \infty, \tag{8}$$

which is an exact representation of the first model selection function F_1. For all other iterations $t > 1$, the algorithm begins by discarding any breakpoints which are no longer necessary, then adds one new breakpoint. In detail, at iteration t, we need to additionally minimize over the new cost function, $f_t(\lambda) = L_t + \lambda t$. To update our breakpoints in the minimum cost, we first need to compute new candidate breakpoints, where this new cost function is equal to the cost of a previously selected model $i \in \{1, \ldots, M_{t-1}\}$, that is,

$$L_t + \lambda t = L_{K_{t-1,i}} + \lambda K_{t-1,i}. \tag{9}$$

The penalty λ for that new candidate breakpoint is therefore

$$c(t, i) = \frac{L_{K_{t-1,i}} - L_t}{t - K_{t-1,i}}. \tag{10}$$

Note that $c(t, i)$ is a penalty value, but it is defined in terms of the coefficients of the loss functions (it is the penalty where the loss functions are equal). As an aside, to modify the algorithm to use a more general regularizer $R(\Theta_k) = r_k$, we need to change the denominator of Equation (10) from $t - K_{t-1,i}$ to $r_t - r_{K_{t-1,i}}$.

The algorithm then computes the largest model size i for which the candidate breakpoint $c(t, i)$ is less than the previously stored breakpoint $b_{t-1,i-1}$,

$$I_t = \max\left\{i \in \{1, \ldots, M_{t-1}\} | c(t, i) < b_{t-1,i-1}\right\}. \tag{11}$$

The model I_t is the largest from the previous iteration which is kept during this iteration (all larger sizes $i > I_t$ are discarded because they are no longer optimal for any penalty). Note that $I_t \geq 1$ for any t, because the inequality in (11) is always true for $i = 1$, that is, $c(t, 1) < b_{t-1,0} = \infty$ for any t.

We then compute the number of models selected at iteration t,

$$M_t = I_t + 1. \tag{12}$$

The algorithm stores the new set of models selected at iteration t,

$$K_{t,i} = \begin{cases} K_{t-1,i} & \text{for } i \in \{1, \ldots, I_t\} \\ t & \text{for } i = M_t. \end{cases} \tag{13}$$

The algorithm also stores the new breakpoint $c(t, I_t)$ along with some of the previous breakpoints,

$$b_{t,i} = \begin{cases} b_{t-1,i} & \text{for } i \in \{0, \ldots, I_t - 1\} \\ c(t, I_t) & \text{for } i = I_t \\ 0 & \text{for } i = M_t. \end{cases} \tag{14}$$

Once the selected models $K_{t,i}$ and breakpoints $b_{t,i}$ have been recursively computed via (13) and (14), the model selection function F_t is defined using Equation (7).

2.3. Demonstration of Algorithm up to $t = 3$

In this section, we provide two example runs of the algorithm. The initialization creates an exact representation of $k^*_1 = F_1$, via one possible model $K_{1,1} = 1$ with loss $L_1 = 7$ which is selected for all λ between the two breakpoints $b_{1,1} = 0 < b_{1,0} = \infty$. The second iteration uses $L_2 = 4$ to compute the candidate breakpoint $c(2, 1) = 3$ which is stored in the new breakpoints $b_{2,2} = 0 < b_{2,1} = c(2, 1) = 3 < b_{2,0} = \infty$, along with $M_2 = 2$ models $K_{2,2} = 2 > K_{2,1} = 1$ (Figure 1, left).

At iteration $t = 3$ we first compute the candidate $c(3, 2)$ and compare it to the stored breakpoint $b_{2,1}$. For a small loss value, for example, $L_3 = 0$ (Figure 1, middle), we have $c(3, 2) = 4 \geq b_{2,1} = 3$ so the previous breakpoint $b_{2,1} = 3$ is removed, and the candidate $c(3, 2) = 4$ is discarded. Next, we check

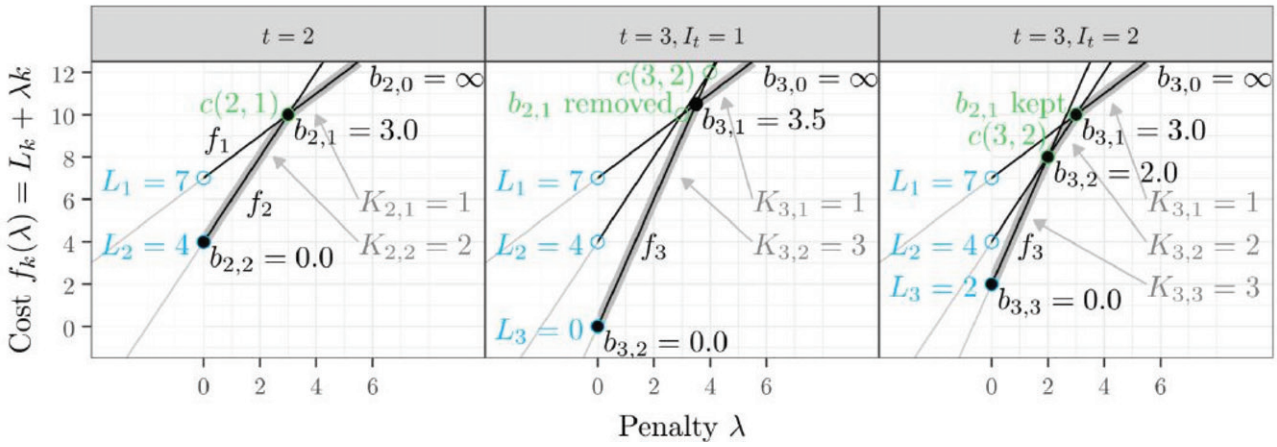

Figure 1. Two possible runs of the algorithm for $N = 3$ models (middle and right panels show how the algorithm works with two different L_3 values, given the same L_1 and L_2 values). *Left:* at iteration $t = 2$, the cost function $f_2(\lambda)$ is minimal (gray curve) for penalties $\lambda < b_{2,1} = c(2,1) = 3$, and $f_1(\lambda)$ is minimal otherwise. *Middle:* at iteration $t = 3$ a small L_3 value results in $c(3,2) = 4 > 3 = b_{2,1}$ so we remove $b_{2,1}$ and $K_{2,2} = 2$, because $f_2(\lambda)$ is no longer minimal for any λ. *Right:* at iteration $t = 3$ a large L_3 value results in $c(3,2) = 2 < 3 = b_{2,1}$ so we keep $b_{2,1} = b_{3,1}$ and store a new breakpoint $c(3,2) = b_{3,2}$.

$c(3,1) = 3.5 < b_{2,0} = \infty$ so $i = 1$ is the largest model satisfying the inequality in (11) and thus $I_3 = 1$. The new breakpoints $b_{3,2} = 0 < b_{3,1} = c(3,1) = 3.5 < \infty = b_{3,0}$ are computed and stored with $M_3 = 2$ models $K_{3,2} = 3 > 1 = K_{3,1}$. For a large loss value, for example, $L_3 = 2$ (Figure 1, right), $c(3,2) = 2 < b_{2,1} = 3$ implies $I_3 = 2$ and the previous breakpoint $b_{2,1} = 3$ is kept along with the candidate $c(3,2) = 2$. The breakpoints $b_{3,3} = 0 < c(3,2) = 2 < b_{2,1} = 3 < b_{3,0} = \infty$ are stored with $M_3 = 3$ models $K_{3,3} = 3 > 2 > 1 = K_{3,1}$.

2.4. Recursive Update Rules Are Optimal

Equations (5)–(14) define a dynamic programming algorithm, because the recursively computed F_N is optimal in the sense of Equation (4), as demonstrated in the following theorem.

Theorem 1 (Update rules yield the optimal model selection function). The recursively computed function F_N (7) and the model selection function k^*_N (4) are identical.

Proof. The proof follows from Equations (5)–(14) using induction on t. The base case is $t = 1$, for which the initialization (8) of the recursively computed function implies $F_1(\lambda) = 1$ for all $\lambda \in (0, \infty)$. Because at iteration $t = 1$, there is only one possible model, it is clear that $F_1(\lambda) = k^*_1(\lambda)$ for all λ.

The proof by induction now assumes that $F_{t-1}(\lambda) = k^*_{t-1}(\lambda)$ for all λ; we will prove that the same is true for t. The recursive updates (13)–(14) imply that

$$F_t(\lambda) = \begin{cases} K_{t-1,1} & \text{if } \lambda \in (b_{t-1,1}, b_{t-1,0}) \\ \quad\vdots & \\ K_{t-1,I_t} & \text{if } \lambda \in (c(t,I_t), b_{t-1,I_t-1}) \\ t & \text{if } \lambda \in (0, c(t,I_t)) \end{cases} \tag{15}$$

$$= \begin{cases} F_{t-1}(\lambda) & \text{if } \lambda > c(t,I_t) \\ t & \text{if } \lambda < c(t,I_t) \end{cases} \tag{16}$$

We need to prove that the function above returns the model size $k \in \{1, \ldots, t\}$ with min cost $f_k(\lambda)$, for any penalty λ. Equations (11)–(10) imply $K_{t-1,I_t} = F_{t-1}[c(t,I_t)]$ is the min cost model at the penalty $c(t,I_t)$ where the new cost function f_t equals the previous min cost function,

$$f_t[c(t,I_t)] = f_{K_{t-1,I_t}}[c(t,I_t)] = \min_{k\in\{1,\ldots,t-1\}} f_k[c(t,I_t)]. \tag{17}$$

Because $f_t(\lambda) = L_t + \lambda t$ is a linear function with a larger slope than any of $f_1, \ldots, f_{t-1}$, and a smaller intercept $L_t < L_{t-1} < \cdots$, we therefore deduce that f_t is less costly before $c(t,I_t)$, and more costly after:

$$\begin{cases} f_t(\lambda) < \min_{k\in\{1,\ldots,t-1\}} f_k(\lambda) & \text{for all } \lambda < c(t,I_t) \\ f_t(\lambda) > \min_{k\in\{1,\ldots,t-1\}} f_k(\lambda) & \text{for all } \lambda > c(t,I_t) \end{cases} \tag{18}$$

Combining Equations (16), (18) and using the induction hypothesis completes the proof that $F_t(\lambda) = k^*_t(\lambda) = \arg\min_{k\in\{1,\ldots,t\}} f_t(\lambda)$ for all λ. □

3. Linear Time Algorithm With Theoretical Complexity Analysis

In this section, we propose pseudocode that efficiently implements the dynamic programming algorithm, and provide a proof of worst-case linear time complexity. We also provide a theoretical analysis of the previous quadratic time algorithm in terms of the framework of this article.

3.1. Proposed Linear Time Algorithm

We propose Algorithm 1, which is pseudocode for Equations (11)–(14). It recursively computes an exact representation of the model selection function F_N in terms of breakpoints b and selected models K.

It begins by initializing the model selection function F_1 (line 3). Then for all $t \in \{2, \ldots, N\}$ it recursively computes F_t from F_{t-1}. The first step in the loop (Line 5) is to call the SOLVE sub-routine, which computes the number of selected models M_t and the new breakpoint $\lambda = c(t, M_t - 1)$. The number of while loop evaluations $w[t]$ can optionally be stored in order to analyze empirical time complexity. The next step is to store the new model t and new breakpoint λ (line 6), which completes the computation of F_t.

Algorithm 1 Dynamic programming for computing exact representation of model selection function

1: Input: Array of N real numbers $L[1] > \cdots > L[N]$ (decreasing loss values).
2: Allocate: selected models $K \in \mathbb{Z}^N$, breakpoints $b \in \mathbb{R}^N$, while loop iterations $w \in \mathbb{Z}^N$
3: Initialize: number of models $M \leftarrow 1$, breakpoint $b[1] \leftarrow \infty$, selected model $K[1] \leftarrow 1$
4: **for** $t = 2$ to N **do**
5: $M, \lambda, w[t] \leftarrow \text{SOLVE}(K, b, L, M, t)$
6: $b[M] \leftarrow \lambda$, $K[M] \leftarrow t$ // store a new breakpoint
7: **end for**
8: Output: selected models $K[1:M]$, breakpoints $b[1:M]$, while loop iterations $w[2:N]$

In this article, we propose an amortized constant $O(1)$ time implementation of the SOLVE sub-routine (Algorithm 2). It computes I_t, M_t by solving the maximization in Equation (11) using a linear search over possible values of the model index i. It starts at the current number of selected models (Line 2), and then repeatedly tests the criterion from Equation (11). If the current value of the model index i does not satisfy the condition of the while loop (Line 3), then the model index is decremented to remove a breakpoint (Line 4). The number of while loop iterations w_t (Lines 2, 4) can be optionally computed in order to analyze the empirical time complexity of the algorithm. Even though Algorithm 2 is clearly $O(M)$ in the worst case, in the next section we prove that it is amortized constant $O(1)$ time when used in the context of Algorithm 1. Using this sub-routine therefore results in an overall linear $O(N)$ time complexity for Algorithm 1, in the best and worst case (Table 2).

Algorithm 2 Proposed SOLVE sub-routine

1: Input: selected model sizes $K \in \mathbb{Z}^N$, breakpoints $b \in \mathbb{R}^N$, loss values $L \in \mathbb{R}^N$, number of selected models $M \in \mathbb{Z}$, new model size $t \in \mathbb{Z}$.
2: $i \leftarrow M$, $w_t \leftarrow 1$
3: **while** $\lambda \leftarrow (L[K[i]] - L[t])/(t - K[i]) \geq b[i]$ **do**
4: $i--$, w_t++ // remove a breakpoint
5: **end while**
6: Output: number of models $i+1$, new breakpoint λ, number of while loop iterations w_t

3.2. Previous Quadratic Algorithms

In this section, we provide a detailed comparison with several previously proposed quadratic algorithms (Arlot and Massart 2009; Hocking et al. 2013). In terms of the framework of this article, these previous algorithms can be interpreted as computing $K_{t,i}, b_{t,i}$ for $t = N$, without computing any of the solutions at the previous iterations $t < N$. These other algorithms are therefore not performing dynamic programming. Whereas our algorithm starts at the smallest model size and then updates the model selection function for larger sizes, these other algorithms begin at the largest model size. In particular they start by initializing the largest model $K_{N,M_N} = N$ and the smallest breakpoint $b_{N,M_N} = 0$, then for all $i \in \{M_N - 1, \ldots, 1\}$ they recursively compute $K_{N,i}, b_{N,i}$ from $K_{N,i+1}, b_{N,i+1}$. There are $M_N - 1$ iterations of this recursive computation, and each iteration considers $K_{N,i+1} - 1$ breakpoints. The overall algorithm is therefore $O(NM_N)$; best case $O(N)$ is when the number of selected models $M_N = 2$ is small; worst case $O(N^2)$ is when $M_N = N$ is large (Table 2). Interestingly, the opposite is true of our algorithm (best case is when M_N is large), as we prove in the next section. We show that even in the worst case for our algorithm (when M_N is small), it is still asymptotically linear $O(N)$. The practical implication is that our algorithm results in big speedups over the previous algorithm when the number of selectable models M_N is large, and has the same linear time complexity when M_N is small.

Table 2. Summary of asymptotic complexity in terms of number of input models, N.

Algorithm	Best time	Worst time	Space
This paper, Algorithm 1	$O(N)$	$O(N)$	$O(N)$
Arlot and Massart (2009) and Hocking et al. (2013)	$O(N)$	$O(N^2)$	$O(N)$

3.3. Proof of Linear Time and Space Complexity

The overall space complexity of Algorithm 1 is clearly $O(N)$, because up to N possible models/breakpoints can be computed. This is the same storage/space complexity as previously proposed algorithms (Table 2).

The time complexity depends on the implementation of the SOLVE sub-routine (line 5). The computation time of our proposed implementation of the SOLVE sub-routine (Algorithm 2) depends on w_t, the number of times the while condition is evaluated (Line 3). In particular, the overall time complexity of Algorithm 1 is linear in total number of times the while condition is checked,

$$W_N = \sum_{t=2}^{N} w_t. \tag{19}$$

The following result proves that Algorithm 1 is overall $O(N)$ time, by bounding the total number of times the while condition is checked.

Theorem 2 (Best and worst-case time complexity). For any N inputs to Algorithm 1, the total number of while loop iterations W_N over all calls to Algorithm 2 is bounded: $N - 1 \leq W_N \leq 2N - 3$.

Proof. The proof uses the fact that for all $t \in \{2, \ldots, N\}$, we have

$$M_t = 2 + M_{t-1} - w_t, \tag{20}$$

which follows from the definition of the number of while loop iterations w_t (on Line 4 of Algorithm 2, every iteration decrements i, and therefore M_t). The total number of while loop iterations is thus

$$W_N = \sum_{t=2}^{N} w_t = \sum_{t=2}^{N} 2 + M_{t-1} - M_t \tag{21}$$

$$= 2(N-1) + \sum_{t=2}^{N} M_{t-1} - \sum_{t=2}^{N} M_t \tag{22}$$

$$= 2(N-1) + \sum_{t=1}^{N-1} M_t - \sum_{t=2}^{N} M_t \tag{23}$$

$$= 2N - 2 + M_1 - M_N \tag{24}$$

$$= 2N - 1 - M_N. \tag{25}$$

The first two equalities (21) follow from the definitions of the number of while loop iterations (19)–(20). The next equalities come from distributing the sum (22), then re-writing the second term as a sum from $t = 1$ to $N-1$ (23). The last equalities come from subtracting the terms in the two sums (24), then using the fact that $M_1 = 1$ (25). The result is obtained using the fact that the number of selectable models is bounded, $2 \leq M_N \leq N$. □

The best case of Algorithm 1, $W_N = N - 1$ iterations, happens when the number of selected models is large, $M_N = N$; the worst case $W_N = 2N - 3$ iterations occurs when $M_N = 2$. Because the total number of iterations is always $O(N)$, the SOLVE sub-routine (Algorithm 2) is amortized constant $O(1)$ time on average, even though it is linear in the number of models $O(M)$ in the worst case.

4. Empirical Complexity Analysis

In this section, we empirically examine the number of iterations of our algorithm, and show that it is overall orders of magnitude faster than previous baselines.

4.1. Empirical Iteration Counts Are Consistent With Theoretical Bounds

As discussed in Section 3.3, the time complexity of Algorithm 1 is linear W_N, the total number of iterations of the while loop in the SOLVE sub-routine. Here, we demonstrate that the theoretical bounds on W_N obtained in Theorem 2 are consistent with the number of iterations obtained empirically in real and synthetic data. First, we considered 1000 real cancer DNA copy number datasets of different sizes $p \in \{2, \ldots, 869\}$ from R package `neuroblastoma`. For each sequence dataset $\mathbf{z} \in \mathbb{R}^p$, we used the Pruned Dynamic Programming Algorithm (PDPA) of Rigaill (2015) to compute a sequence of optimal changepoint models. For each number of segments $k \in \{1, \ldots, p\}$, the optimal loss is

$$L_k = \min_{\theta \in \mathbb{R}^p} \sum_{j=1}^{p} (\theta_j - z_j)^2 \tag{26}$$

$$\text{subject to } ||D\theta||_0 = \sum_{j=1}^{p-1} I[\theta_j \neq \theta_{j+1}] = k - 1.$$

The PDPA returns a regularization path of $N = p$ models, from $k = 1$ segment (no changepoints, $\theta_j = \theta_{j+1}$ for all j) to $k = N = p$ segments (change after every data point, $\theta_j \neq \theta_{j+1}$ for all j). We used the resulting loss values $L_1 > \cdots > L_N$ as input to Algorithm 1. We plotted the number of iterations W_N as a function of dataset size N (black points in Figure 2), and observed that they always fall between the upper/lower bounds from Theorem 2 (gray lines). These results provide empirical evidence that the time complexity of our algorithm is linear $O(N)$ in real data. Furthermore, we observed that the number of iterations in these real data tends to be closer to the lower bound than to the upper bound. Because the number of iterations depends on the distribution of loss L_k values, this result suggests that there are a relatively large number of selectable models M_N in these real data. More generally, we expect results near the upper/lower bounds for datasets with few/many selectable models M_N (e.g., few/many significant changepoints with corresponding significant jumps in loss values).

Second, we considered two synthetic sequences of loss values, $L_t = N - t$ for all $t \in \{1, \ldots, N\}$ (e.g., $L_1 = 4 > 3 > 2 > 1 > 0 = L_5$ for $N = 5$) and $L_t = N - \sqrt{t}$ (e.g., $L_1 = 4 > 3.59 > 3.27 > 3 > 2.76 = L_5$ for $N = 5$). For these loss values, we observed a number of iterations (violet points in Figure 2) that always falls on the upper/lower bounds (gray lines), which indicates that these synthetic data achieve the worst/best case. Overall these results provide a convincing empirical validation of our theoretical bounds from Theorem 2.

4.2. Empirical Timings Suggest Orders of Magnitude Speedups

Our proposed algorithm takes as input a sequence of N loss values, which must be computed by some other machine learning algorithm. In this section, we therefore analyzed our algorithm

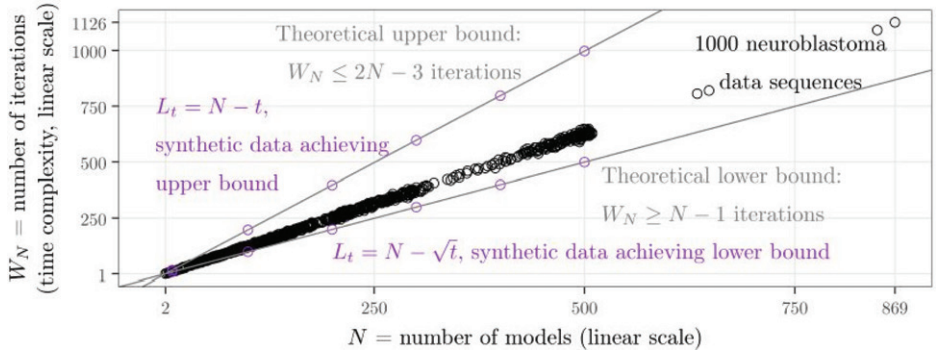

Figure 2. Empirical verification of the theoretical bounds on the total number of iterations W_N during a single call to our proposed Algorithm 1. In synthetic loss values (light points) and loss values from optimal changepoint models of real neuroblastoma data (dark points), the total number of iterations is $W_N = O(N)$, linear in the number of input models N, consistent with theoretical upper/lower bounds obtained in Theorem 2 (gray lines).

in the context of a two-step pipeline: (i) compute the N loss values, (ii) compute an exact representation of the model selection function. The overall time complexity of the two-step pipeline is determined by the slower of the two steps. If the first step is at least quadratic, then the pipeline is as well (using either linear or quadratic time model selection in the second step). However, if the first step is sub-quadratic, then we expect that our linear time algorithm in the second step should result in speedups.

4.2.1. Simulated Data for Which Proposed Linear Time Algorithm Results in Speedups

For the first step, we therefore use the log-linear binary segmentation algorithm, which inputs a data sequence $\mathbf{z} \in \mathbb{R}^P$, and computes an approximate solution to (26). The binary segmentation algorithm computes the full path of $N = p$ models with corresponding loss values $L_1, \ldots, L_N$ in $O(N \log N)$ time on average (Scott and Knott 1974; Truong, Oudre, and Vayatis 2018). For each dataset size $N \in \{10^2, \ldots, 10^5\}$, we generate synthetic data sequences $z_j = \sin(j) + j/N$, for all $j \in \{1, \ldots, N\}$. Note that these synthetic data are not meant to be typical or representative changepoint problems; instead they are meant to illustrate a situation for which our linear time model selection algorithm results in speedups over the previous quadratic time algorithm. Figure 3 (left) shows timings of binary segmentation alone (binseg), exact model selection algorithms alone (linear, quadratic), and two-step pipelines (binseg.linear, binseg.quadratic), on an Intel T2390 1.86GHz CPU. As expected, our proposed linear time algorithm is orders of magnitude faster than the previous quadratic time algorithm (when run alone, and also in the two-step pipeline). For example, for $N = 10^5$ data, the binseg.linear pipeline takes about 3 seconds, whereas binseg.quadratic takes about 2 min. More generally, such timings are typical for any data for which binary segmentation runs in log-linear time, and the number of selected models M_N increases with the dataset size N (second column of Figure 4i).

4.2.2. Simulated Data for Which Proposed Linear Time Algorithm Results in No Speedups

However, there are other kinds of data for which our approach are no faster than the quadratic baseline (other columns of Figure 4). For example, when binary segmentation runs in quadratic time, then our linear time model selection algorithm offers no speedups to the overall pipeline (third and fourth columns of Figure 4). Also, since the previous (worst-case quadratic) algorithm achieves its best case linear time complexity when the number of selected models M_N is small/constant, then our proposed algorithm offers no speedups in this case (first columns of Figure 4). Overall, we have shown that for some datasets, our linear time algorithm provides substantial speedups relative to the previous quadratic time algorithm.

4.2.3. Real Data for Which Proposed Linear Algorithm is Faster Than Grid Search

Another baseline algorithm for computing a representation of the model selection function is a naïve approximate grid search over G penalties λ, which takes $O(NG)$ time. We expected this baseline to perform poorly in the context of large N and large G, so we performed timings on a large `chipseq` dataset from the UCI repository (Newman and Merz 1998). We first computed a regularization path of $N = 287{,}443$ optimal changepoint models for a sequence of $p = 1{,}656{,}457$ data, and then performed timings of the model selection algorithms on the resulting N loss values. We observed that our proposed linear time algorithm is always faster than approximate grid search with at least 10 grid points (Figure 3, right). For example, the approximate grid search takes almost 2 min for $N = 10{,}000$ grid points, whereas the proposed exact linear time algorithm takes only 27 milliseconds. Overall these data indicate that the proposed linear time algorithm is indeed faster than the two baselines in large data.

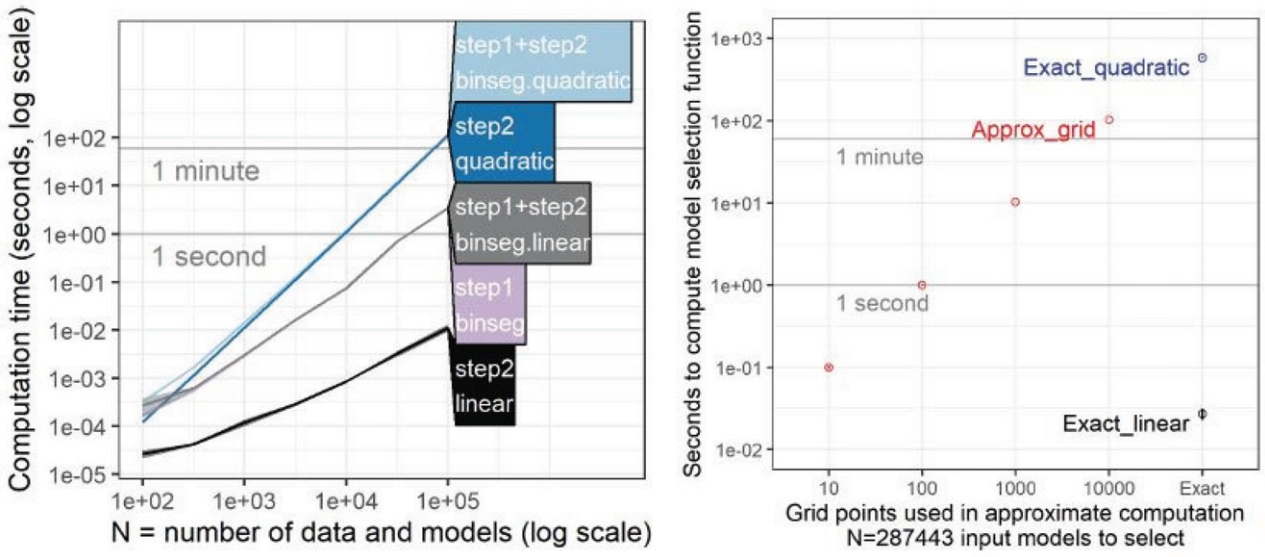

Figure 3. The proposed exact linear time algorithm is orders of magnitude faster than the previous exact quadratic time algorithm and naïve approximate grid search. *Left:* the two exact algorithms compute the same result, but the proposed linear time algorithm is orders of magnitude faster, even when time to compute loss values via binary segmentation (binseg) is included in the timing (lines/bands for mean/SD over 5 timings). *Right:* when used on loss values from $N = 28,7443$ optimal changepoint models for one genomic data sequence, the proposed exact linear time algorithm is always faster than approximate grid search with at least 10 points (points/segments for mean/SD over 5 timings).

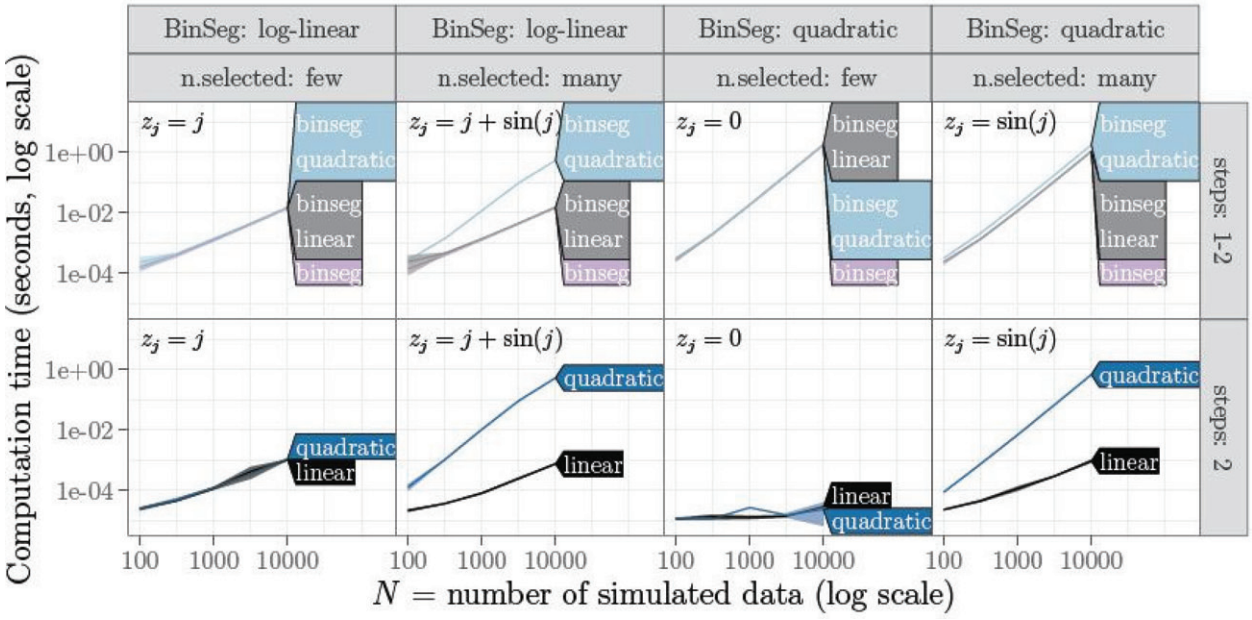

Figure 4. Binary segmentation (Step 1) followed by exact model selection (Step 2) was run on four synthetic data sequences (panels from left to right). Both model selection algorithms (worst-case quadratic, and proposed worst-case linear time) output the exact path of selected models. *Bottom:* when there are few selected models (first and third columns) the quadratic algorithm achieves its best case linear time complexity; when there are many selected models (second and fourth columns) it achieves the worst-case quadratic time complexity. *Top:* total timings over both steps show that the linear time algorithm offers substantial speedups when binary segementation achieves its best-case log-linear time complexity, and there are many selected models (second column).

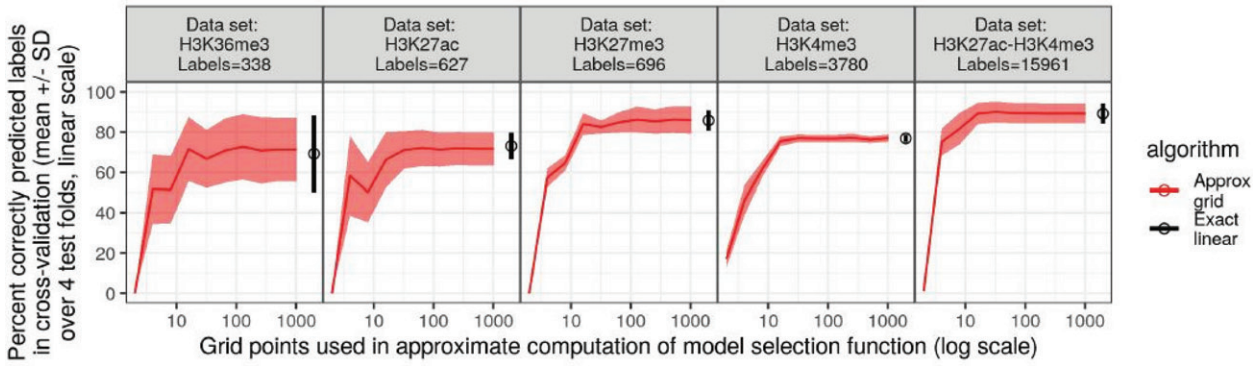

Figure 5. Four-fold cross-validation to measure prediction accuracy on held-out test data increases as a function of number of points used in approximate grid search algorithm (red line/band); it takes 10–100 grid points to achieve the maximum accuracy in each dataset (panels), which is also achieved by the proposed linear time exact algorithm (black point/error bar on right).

5. Real Data Applications and Comparisons

5.1. *Prediction Accuracy in Supervised Changepoint Problems*

In this section, we aim to demonstrate that the proposed exact algorithm results in more accurate predictions than a naïve approximate grid search. To examine the accuracy of our algorithm, we consider several supervised changepoint detection problems from the UCI `chipseq` data (https://archive.ics.uci.edu/ml/datasets/chipseq), which contain labels that indicate the presence/absence of changepoints in particular data subsets. Accurate changepoint detection in these data is important in order to characterize active/inactive regions in the human epigenome.

Here, we give a brief summary of the supervised learning framework for changepoint detection; for details, see Hocking et al. (2013). Each observation i is represented by a numeric data vector/sequence $\mathbf{z}_i$ along with a corresponding label set ℓ_i. We compute a feature vector $\mathbf{x}_i$ then learn a penalty function $f(\mathbf{x}_i) = \log \lambda_i$ which results in a model $\hat{\theta}(\lambda_i)$. The goal is to learn a function f that results in minimal errors with respect to the labels ℓ_i in test data sequences. In this context there is a model selection function k_i^* which is specific to each data sequence i, and is used in two places during the learning and prediction. First, it is used to compute a learning target y_i which is an interval of optimal penalty values for each training data sequence i. Second, it is used to compute the predicted model $\hat{\theta}(\lambda_i)$ given a predicted penalty λ_i. We learn a linear penalty function f as previously described (Hocking et al. 2013), using intervals y_i computed by either our exact algorithm or a naïve approximate grid search with a variable number of penalties λ.

We performed 4-fold cross-validation in five different labeled datasets (panels in Figure 5). We observed in each dataset that it takes 10–100 penalties λ in the grid search to achieve the

maximum number of correctly predicted labels, which was also achieved by the proposed exact algorithm. Overall these data provide empirical evidence that, in the context of supervised changepoint detection problems, using an exact representation of the model selection function results in more accurate predictions than using an approximate representation obtained via grid search.

5.2. Speed Comparison With Previous CROPS Algorithm

The goal of this section is to demonstrate that the proposed algorithm achieves similar time complexity to the existing CROPS algorithm (Haynes, Eckley, and Fearnhead 2017), in the context of optimal changepoint detection. As mentioned in Section 1.1, our proposed algorithm should yield the same result as the previous CROPS algorithm when computing the full path of optimal changepoint models for a given data sequence. We used 10 real cancer copy number data sequences from R package `neuroblastoma`, ranging in size from $N = 2$ to $N = 5937$ data. In these data there are abrupt changes in mean, so we used a normal model with change in mean and constant variance. For each dataset, we computed timings for three algorithms: (i) CROPS with PELT as implemented in `changepoint::cpt.mean` in R, (ii) PDPA as implemented in `jointseg::Fpsn` in R, and (iii) our proposed exact linear time model selection algorithm. We expected that for computing the full path of models, PDPA combined with model selection should have similar asymptotic timings to the CROPS with PELT approach. Consistent with these expectations, we observed that the computation times of CROPS and PDPA are asymptotically similar, with PDPA being slightly faster by constant factors (Figure 6). We furthermore observed that our proposed model selection algorithm, when applied to the loss values computed using the PDPA, was much faster than the other algorithms for sequences with at least 100 data. Overall these data suggest that our proposed approach using PDPA with linear time model selection is comparable to CROPS in terms of empirical time complexity.

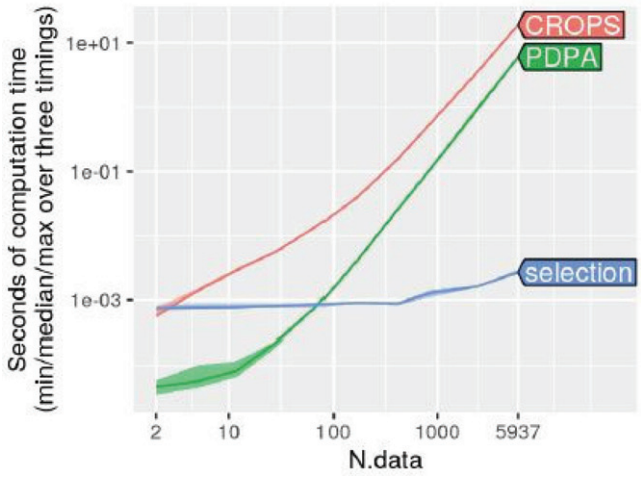

Figure 6. Computation time of proposed linear time model selection algorithm and PDPA/CROPS algorithms for computing optimal changepoint models.

5.3. Applications to Other Problems

In this section, the goal was to show the applicability of our proposed algorithm to problems other than changepoint detection (Table 1). The first example problem was clustering, for which used the K-means algorithm to compute a sequence of models between $k = 1$ and 200 clusters on the zip test data[1] (these are 2007 images of handwritten digits, each with 256 features/pixels, and there are ten classes). We used the total square loss between data points and cluster centers as the loss L_k input to our model selection algorithm (Figure 7, left). The second example problem was dimensionality reduction, for which we used principal components analysis (PCA) to compute a sequence of models from $k = 0$ to 256 components on the zip test data, then we used the total square loss between the data and model predictions as the loss L_k input to our model selection algorithm (Figure 7, center). The third example problem was best subset regression, for which we used logistic regression on the spam data (4601 observations, 57 features, binary classification). We used greedy forward stepwise regression, starting with none of the 57 features, then at each step adding the feature which resulted in the best logistic loss decrease on the train set. This resulted in a sequence of models from $k = 0$ to 57 features, and we used the total logistic loss, L_k, as input to our model selection algorithm (Figure 7, right). In each example our proposed model selection algorithm returned a model selection function that maps penalty values $\lambda \geq 0$ to model sizes k. In two of the problems (K-means and logistic regression), the model selection resulted in discarding some model sizes k which were not optimal for any penalty values λ, and in one problem (PCA) the result was that all model sizes k were selected for at least one penalty λ value. Overall these data show that our proposed algorithm can be applied to various different learning problems in order to compute the exact model selection function.

6. Discussion and Conclusions

For learning problems with ℓ_0 regularization, we proposed a new dynamic programming algorithm for computing an exact representation of the model selection function (4). By bounding the number of iterations, we proved theoretically that the algorithm is linear time in the worst case. In real and synthetic data, we empirically validated these bounds, and showed that the proposed linear time algorithm is orders of magnitude faster than two baselines. We used cross-validation in supervised changepoint detection problems to show that the exact representation provides more accurate predictions than the grid search approximation baseline.

Our algorithm requires no special data structures and can be efficiently implemented using arrays in standard C; our free software implementation is available at *https://github.com/tdhock/penaltyLearning/* and as an R package with function `modelSelection` (R Core Team 2021).

For reproducibility, we also provide the source code that we used to make the figures in this article at *https://github.com/tdhock/changepoint-data-structure*. For future work, we

[1] zip and spam data in this section from *https://web.stanford.edu/~hastie/ElemStatLearn/*

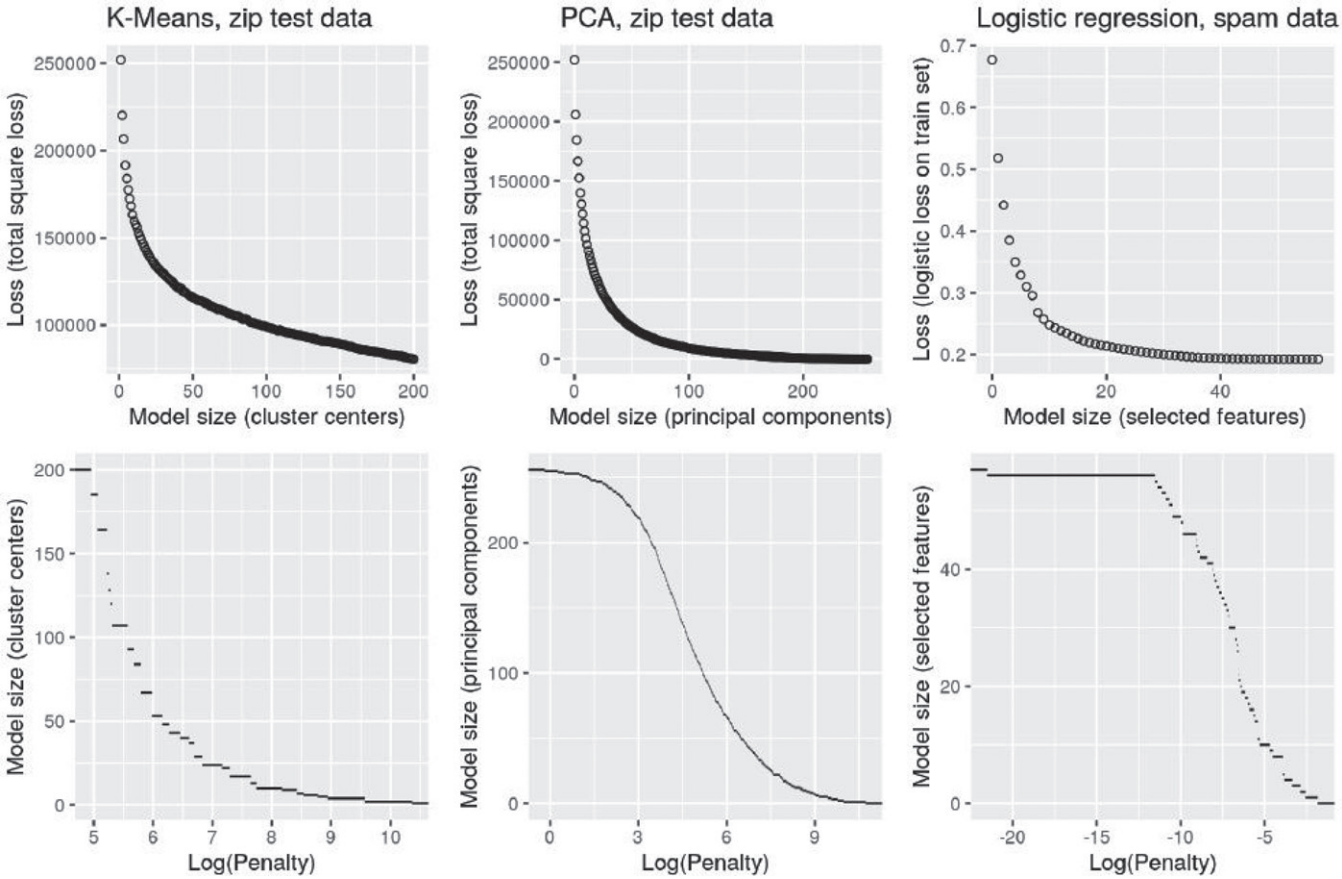

Figure 7. Results of computing exact model selection function on three different algorithms and datasets (panels from left to right). *Top:* each algorithm returns a sequence of models each with a corresponding loss value L_k for model size k. *Bottom:* the exact model selection function is the selected model size as a function of the (log) penalty.

would like to consider selecting models from a partial set $S \subset \{1, \ldots, N\}$, and develop an efficient algorithm for updating an exact representation of the corresponding model selection function.

References

Arlot, S., and Massart, P. (2009), "Data-Driven Calibration of Penalties for Least-Squares Regression," *Journal of Machine Learning Research*, 10, 245–279. [314,317]

Auger, I., and Lawrence, C. (1989), "Algorithms for the Optimal Identification of Segment Neighborhoods," *Bulletin of Mathematical Biology*, 51, 39–54. [313]

Bellman, R. (1961), "On the Approximation of Curves by Line Segments Using Dynamic Programming," *Communications of the ACM*, 4, 284. [315]

Bertsimas, D., King, A., and Mazumder, R. (2016), "Best Subset Selection Via a Modern Optimization Lens," *The Annals of Statistics*, 44, 813–852. [313]

Chen, S., Donoho, D., and Saunders, M. (1998), "Atomic Decomposition by Basis Pursuit," *SIAM Journal on Scientific Computing*, 20, 33–61. [313]

Davis, G., Mallat, S., and Zhang, Z. (1994), "Adaptive Time-Frequency Decompositions With Matching Pursuit," *Wavelet Applications*, 402, 402–413. [313]

Fan, J., and Li, R. (2001), "Variable Selection Via Nonconcave Penalized Likelihood and Its Oracle Properties," *Journal of the American Statistical Association*, 96, 1348–1360. [313]

Haynes, K., Eckley, I. A., and Fearnhead, P. (2017), "Computationally Efficient Changepoint Detection for a Range of Penalties," *Journal of Computational and Graphical Statistics*, 26, 134–143. [313,314,321]

Huang, S., and Wolkowicz, H. (2018), "Low-Rank Matrix Completion Using Nuclear Norm Minimization and Facial Reduction," *Journal of Global Optimization*, 72, 5–26. [313]

Jackson, B., Scargle, J. D., Barnes, D., Arabhi, S., Alt, A., Gioumousis, P., Gwin, E., Sangtrakulcharoen, P., Tan, L., and Tsai, T. T. (2005), "An Algorithm for Optimal Partitioning of Data on an Interval," *IEEE Signal Processing Letters*, 12, 105–108. [313]

Killick, R., Fearnhead, P., and Eckley, I. A. (2012), "Optimal Detection of Changepoints With a Linear Computational Cost," *Journal of the American Statistical Association*, 107, 1590–1598. [314]

Lavielle, M. (2005), "Using Penalized Contrasts for the Change-Point Problem," *Signal Processing*, 85, 1501–1510. [314]

MacQueen, J. (1967), "Some Methods for Classification and Analysis of Multivariate Observations," in *Proc. of the Fifth Berkeley Symp. on Math. Stat. and Prob.*, pp. 281–297. [313]

Maidstone, R., Hocking, T., Rigaill, G., and Fearnhead, P. (2016), "On Optimal Multiple Changepoint Algorithms for Large Data," *Statistics and Computing*, 27, 519–533. [314]

Mallat, S., and Zhang, Z. (1993), "Matching Pursuits With Time-Frequency Dictionaries," *IEEE Transactions on Signal Processing*, 41, 3397–3415. [313]

Mazumder, R., Friedman, J., and Hastie, T. (2011), "Sparsenet: Coordinate Descent With Nonconvex Penalties," *Journal of the American Statistical Association*, 106, 1125–1138. [313]

Miller, A. (2002), *Subset Selection in Regression* (2nd ed.), Chapman and Hall. [313]

Newman, C. L., Blake, D. J., and Merz, C. J. (1998), *UCI Repository of machine learning databases*. University of California, Irvine, Dept. of Information and Computer Sciences. [319]

PEGWiki. (2018), "Convex Hull Trick." Available at: *https://wcipeg.com/wiki/Convex_hull_trick*. [314]

Hocking, T., Rigaill, G., Vert, J.-P., and Bach, F. (2013), "Learning Sparse Penalties for Change-Point Detection Using Max Margin Interval Regression," in *Proc. 30th ICML*, pp. 172–180. [314,317,320]

R Core Team. (2021), *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. Available at: *https://www.R-project.org/*. [321]

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, 180–205. [318]

Schniter, P., Potter, L. C., and Ziniel, J. (2009), "Fast Bayesian Matching Pursuit: Model Uncertainty and Parameter Estimation for Sparse Linear Models," *IEEE Transactions on Signal Processing*, 1–12. [313]

Scott, A., and Knott, M. (1974), "A Cluster Analysis Method for Grouping Means in the Analysis of Variance," *Biometrics*, 30, 507–512. [314,319]

Soussen, C., Idier, J., Brie, D., and Duan, J. (2010), "From Bernoulli-Gaussian Deconvolution to Sparse Signal Restoration," Technical Report, 34pp. Available at: *https://hal.archives-ouvertes.fr/hal-00443842*. [313]

Tibshirani, R. (1996), "Regression Shrinkage and Selection Via the Lasso," *Journal of Royal Statistical Society*, Series B, 58, 267–288. [313]

Truong, C., Oudre, L., and Vayatis, N. (2018), "A Review of Change Point Detection Methods," arXiv:1801.00718. [314,319]

van den Burg, G. J. J., Groenen, P. J. F., and Alfons, A. (2017), "SparseStep: Approximating the Counting Norm for Sparse Regularization," arXiv: 1701.06967. [313]

Zhang, C.-H. (2010), "Nearly Unbiased Variable Selection Under Minimax Concave Penalty," *Annals of Statistics*, 38, 894–942. [313]