

Supplementary materials for Seeded Binary Segmentation: A general methodology for fast and optimal change point detection

BY S. KOVÁCS

*Seminar for Statistics, Department of Mathematics, ETH Zurich,
Rämistrasse 101, 8092 Zurich, Switzerland*
solt.kovacs@stat.math.ethz.ch

5

H. LI

*Institute for Mathematical Stochastics, and MBExC Cluster of Excellence (Multiscale
Bioimaging: from Molecular Machine to Networks of Excitable Cells), University of Göttingen,
Goldschmidtstrasse 7, 37077 Göttingen, Germany*
housen.li@mathematik.uni-goettingen.de

10

P. BÜHLMANN

*Seminar for Statistics, Department of Mathematics, ETH Zurich,
Rämistrasse 101, 8092 Zurich, Switzerland*
peter.buehlmann@stat.math.ethz.ch

15

A. MUNK

*Institute for Mathematical Stochastics, and MBExC Cluster of Excellence (Multiscale
Bioimaging: from Molecular Machine to Networks of Excitable Cells), University of Göttingen,
Goldschmidtstrasse 7, 37077 Göttingen, Germany*
munk@math.uni-goettingen.de

20

A. RELATED METHODS BASED ON BINARY SEGMENTATION

Binary segmentation (Vostrikova, 1981) works by first finding the best split $\hat{s}_{(0,T]}$ in all data and if $|T_{(0,T]}(\hat{s}_{(0,T]})|$ is above a predefined threshold, then searches for other split points on the left and right segments $(0, \hat{s}_{(0,T]})$ and $(\hat{s}_{(0,T]}, T]$. The same iterative search is continued as long as the gains are still above the predefined threshold and there is a sufficient number of observations in the resulting segments.

25

The wild binary segmentation method (Fryzlewicz, 2014) and the narrowest over threshold method (Baranowski et al., 2019) both generate M random intervals $I_m = (\ell_m, r_m]$, $m = 1, \dots, M$, where the start and end points are drawn independently, uniformly and with replacement from $\{0, 1, \dots, T\}$, possibly subject to some minimal length constraint to be able to evaluate the chosen test statistic. One determines candidate splits and corresponding maximal gains for each of the random intervals via the CUSUM statistics (or some other statistic suitable to the change point problem at hand). The two methods are slightly different beyond this point, in terms of the way the final change point estimates are selected.

30

Wild binary segmentation orders the candidates according to the respective maximal gains (i.e. values of the test statistic) and selects greedily, i.e. the split point with highest value is the first change point estimate. Then all intervals are eliminated from the list which contain this particular change point. From the remaining candidates, once again the one with maximal gain is selected, followed by the elimination of all intervals that contain this second one. The procedure is repeated as long as the split with highest

35

gain remains above a certain predefined threshold. A suitable threshold can be derived from asymptotic theory for example. Out of several possible segmentations corresponding to different thresholds, one can also select the one optimising a chosen information criterion (e.g. proposed by Zhang & Siegmund, 2007).

The narrowest over threshold method selects change points amongst the candidates that have a maximal gain above a predefined threshold. Of these candidates, the one having the shortest (i.e. narrowest) background interval is selected first. Then all intervals that contain this first estimate are eliminated and repeatedly from the left over ones above the predefined threshold the next one with the currently shortest background is selected. This then continues as long as no more maximal gain is above the predefined threshold. It is also possible to repeat this procedure with differing thresholds and then take the one optimising some information criterion. Note that the narrowest over threshold way of selecting the final estimates makes this applicable in more general scenarios, where the model is e.g. piecewise linear.

Elimination of the intervals in each step is necessary for both methods in order to avoid detecting the same change point at slightly different locations for several different background intervals. The two methods allow for detection in a larger class of signals compared to binary segmentation. The intuition behind this is that some of the random intervals will contain only a single change point sufficiently well separated from the boundaries and thus serve as good local backgrounds in which change points are easily detectable. By way of comparison, binary segmentation performs particularly badly in the presence of multiple change points, if the underlying signal exhibits up and down jumps cancelling each other (see Example 1 and Fig. 1 in the main paper).

The main drawback of wild binary segmentation and the narrowest over threshold method is the increased computational cost. This grows linearly with the total length of random intervals used. Note that random intervals have an expected length of $O(T)$. In scenarios with some short spacings between change points, e.g. in frequent change point scenarios, one should draw a higher number of random intervals in order to ensure that there are still sufficiently many intervals with a single change point in them. In the worst case, when drawing essentially all possible intervals, the complexity is cubic, which is clearly not satisfactory for large scale change point detection problems. It is also an issue that one might not know how many change points to expect and thus how many intervals to draw. Fryzlewicz (2020a) summarises these issues as the lack of computational adaptivity of wild binary segmentation and proposes a modification, called wild binary segmentation 2, which is a hybrid of wild and classical binary segmentation in the following sense. In order to speed up, he suggests drawing a smaller number $M' \ll M$ of random intervals, localising the first candidate greedily, and in the resulting left and right segments draw M' random intervals again, localise again, and repeat iteratively. From the resulting preliminary list of candidates the final estimates are again determined based on certain maximal gains. However, this proposal is fairly specific to the univariate Gaussian setup and might fail when adapting to high-dimensional or other more complicated scenarios such as a model that is piecewise linear. We refer to our discussion in Kovács et al. (2020a) for a comparison between wild binary segmentation 2 and our seeded binary segmentation. An interesting two-stage procedure also targeting computational issues for the univariate Gaussian case was proposed by Lu et al. (2017). They obtain pilot estimates of change points on a sparse subsample and then refine these estimates in a second step in the respective neighbourhoods using dense sampling. This can reduce the computational cost to sub-linear, and is thus useful for extremely long time series. However, note that computational issues in high-dimensional setups primarily arise from the fact that data sets are too ‘wide’ and not because of their being ‘long’. Moreover, there is a price to pay in terms of detectable changes using this procedure, namely, being statistically suboptimal.

B. COMPUTATIONAL COSTS

B.1. Time benchmarking

We consider an example without any change points for the purpose of time benchmarking:

Example B.1. Let $X_1, \dots, X_T \sim \mathcal{N}(0, 1)$ be independent observations.

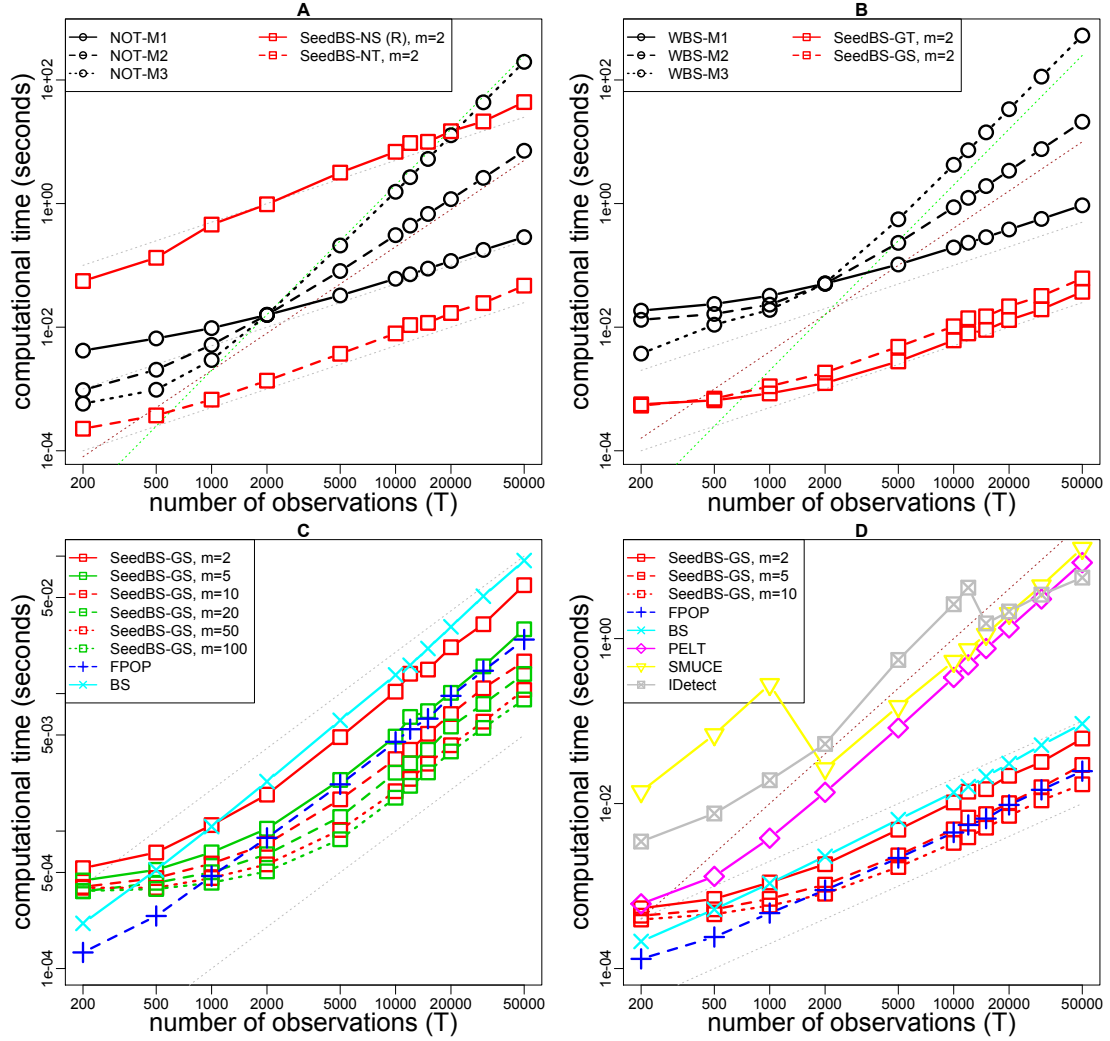


Fig. B.1: Computational times for Example B.1. For seeded binary segmentation, the narrowest over threshold solution path (SeedBS-NS (R)) is implemented in R, the other variants have a faster Fortran based implementation. The minimal segment lengths m used are indicated in the legend and the decay $a = 1/2^{1/2}$ was fixed. Wild binary segmentation (WBS) and the narrowest over threshold method (NOT) use $M1 = 5000$, $M2 = 5000(T/2000)$, $M3 = 5000(T/2000)^2$ random intervals. Other methods shown are binary segmentation (BS); pruned exact linear time (PELT); functional pruning optimal partitioning (FPOP); simultaneous multiscale change point estimator (SMUCE); isolate-detect (IDetect) methods. The thin dashed grey, brown and green lines indicate computational times with scaling $O(T)$, $O(T^2)$ and $O(T^3)$, respectively. Note the logarithmic scales.

As seen on the top two plots (A and B) of Fig. B.1, seeded binary segmentation combined with any greedy (GT and GS) or narrowest over threshold (NT and NS) type selection seems to scale near-linearly. For GT, NT and GS this confirms Theorem 1. For the narrowest over threshold selection even similar thresholds sometimes lead to solutions differing both with respect to number and location of the change point estimates. This makes a tight theoretical analysis of computational cost for the solution path based variant (NS) difficult. However, in practice, close-by thresholds typically produce very similar models, and this can be exploited for faster solution path algorithms rather than naively computing from scratch for each threshold. Some useful tools for this are described in Section G, in particular Remark 2. In the

implementation of the narrowest over threshold selection with full solution path (NS) shown in plot A, we reuse parts of previous segmentations when updating the solution from one threshold to the next. This approach seems to scale near-linearly as well, albeit with a much higher constant, given that the current implementation of this is in $\mathbb{R}$ rather than Fortran as for GT, GS and NT. The computational times for wild binary segmentation and the narrowest over threshold method depend on the chosen number of random intervals and might scale cubic if the number of random intervals is of quadratic order. Plot C in Fig. B.1 compares seeded binary segmentation with full greedy solution path with the fastest competitors, namely, plain binary segmentation and the functional pruning optimal partitioning method. In this example, if the minimal segment length is above 5, then for the default decay of $a = 1/2^{1/2}$, our Fortran based implementation is even somewhat faster than functional pruning. Last, but not least, plot D in Fig. B.1 shows that many other competitors, e.g. the pruned exact linear time method (PELT, Killick et al., 2012); the simultaneous multiscale change point estimator (SMUCE, Frick et al., 2014) and the isolate-detect (IDetect, Anastasiou & Fryzlewicz, 2022) method all can have quadratic run time. Scenarios with quadratic run time are also possible for the functional pruning optimal partitioning method (FPOP, Maidstone et al., 2017) and binary segmentation (BS), however, these scenarios are rather special (see Section B.3).

B.2. Options for speedups

As shown in plot C of Fig. B.1, limiting the minimal segment length can lead to speedups, but care needs to be taken not to choose a value which is too large, otherwise estimation performance could suffer. Naturally, the decay parameter a also has an influence. Here, we recommend choosing at least $a = 1/2^{1/2}$. From a practical perspective, thresholds $\kappa = C \log^{1/2} T$ are of interest for some suitable constant C . Thus, even for the full solution path based variants (GS and NS), one could stop computing segmentations for thresholds that are even smaller than this. In our implementation, one can provide such a threshold to stop at instead of investigating all possible thresholds. This leads to even faster computations for solution path variants (GS and NS) than the ones reported in Fig. B.1. Last, but not least, computations in parallel could be implemented easily for the calculation of the test statistic in the various search intervals, and also for certain selection criteria.

B.3. Worst case computational complexities of competitors

Wild binary segmentation and the narrowest over threshold method can have cubic cost, while the following methods can all have a quadratic computational cost: functional pruning optimal partitioning (for staircase-like signals); binary segmentation (for teeth-like signals, see Example 1 in the rightmost panel of Fig. 1 in the main paper); the isolate-detect method as well as the pruned exact linear time method (in the presence of few change points only as in Example 2 in the main paper or Example B.1 above); or the simultaneous multiscale change point estimator (e.g. Example B.1).

C. DETAILS OF STATISTICAL THEORY

Seeded intervals not only accelerate the computation, but also simplify the statistical analysis compared to random intervals. Consistency results for seeded binary segmentation depend on the assumed model as well as the selection method. As an illustration, we consider again the Gaussian change in mean model for which wild binary segmentation (Fryzlewicz, 2014) and the narrowest over threshold method (Baranowski et al., 2019) were claimed to be minimax optimal. The proof of the former, however, seems to contain errors, e.g., some bounds for the single change point case were wrongly applied to the multiple change point cases. This has been also noted by Wang et al. (2020) in the (sub)Gaussian setup. Contrary to what is claimed by Fryzlewicz (2014), the greedy selection utilised in wild binary segmentation only leads to optimal detection with additional assumptions on the generated intervals or slight modifications of the selection method itself. Otherwise consistency seems only achievable for a smaller class of signals than claimed originally. This is also confirmed by Fryzlewicz (2020b, page 1100), the inventor of wild binary segmentation. Note that wild binary segmentation 2 of Fryzlewicz (2020a) would suffer similarly, as the steepest drop to low levels selection utilised therein is also a certain greedy selection method.

Albeit in a nonparametric setup, Padilla et al. (2021) mention the errors of the original proofs of Fryzlewicz (2014) as well, along some discussion on an additional assumption that they impose under which nearly optimal results can be attained, see their second remark after Theorem 2. However, their assumption is a restriction on the generated search intervals requiring knowledge of the minimal segment lengths. Depending on the data under investigation, such background knowledge may or may not be a realistic assumption. 145

However, the examples we are aware of in which greedy selection could fail are rather special and its empirical performance is often at least as good as e.g. the narrowest over threshold method (see Section E). Some of its other advantages in practice are easy applicability and implementation in various scenarios, that segmentations are nested and stable across thresholds and fast computations is guaranteed. As seeded binary segmentation would suffer similarly if the greedy selection method is chosen, we show the strong consistency using the narrowest over threshold selection method (Baranowski et al., 2019) as an illustration. The following two theorems are more precise versions of Theorem 2 in the main paper. 150

THEOREM C.1. Assume model (1) from the main paper and let $\delta_* = \min_{i=1,\dots,N} \delta_i$ with $\delta_i = |f_{\eta_{i+1}} - f_{\eta_i}|$, and $\lambda = \min_{i=0,\dots,N} |\eta_{i+1} - \eta_i|$ with $\eta_0 = 0$ and $\eta_{N+1} = T$. Assume also that 155

$$\delta_*^2 \lambda \geq C_0 \log T \quad \text{for some constant } C_0. \quad (1)$$

Let $\hat{N}$ and $\hat{\eta}_1 < \dots < \hat{\eta}_{\hat{N}}$ denote respectively the number and locations of estimated change points by seeded binary segmentation with the narrowest over threshold as the selection method. Then there exist constants C_1, C_2 , independent of T and the decay a for seeded intervals, such that, given the threshold for the selection method $\kappa = C_1 \log^{1/2} T$, as $T \rightarrow \infty$,

$$pr \left\{ \hat{N} = N, \max_{i=1,\dots,N} \delta_i^2 |\hat{\eta}_i - \eta_i| \leq C_2 \log T \right\} \rightarrow 1. \quad (2)$$

The signal f , as well as δ_i, δ_* and λ , are allowed to depend on T . The assumption for detection in (1) and the localisation rates $\delta_i^{-2} \log T$ in (2) are minimax optimal up to constants (see e.g. Chan & Walther, 2013; Frick et al., 2014; Chan & Chen, 2017; Li et al., 2019; Verzelen et al., 2020; Wang et al., 2020). That is, no method can detect all N change points with weaker condition than (1), and no method can achieve a better rate than $\delta_i^{-2} \log T$, in the asymptotic sense. It is worth noting that the minimax optimal consistency rates here can also be achieved by dynamic programming based segmentation methods, as briefly discussed in the Introduction of the main paper, see e.g. Lebarbier (2005), Boysen et al. (2009), Du et al. (2016), Garreau & Arlot (2018), Li et al. (2019), Zheng et al. (2022), Wang et al. (2020), Verzelen et al. (2020) for further references. 160

The consistency of seeded binary segmentation can be further established using a certain data adaptive selection rule, i.e., the strengthened Schwarz information criterion, to choose the threshold leading to the best segmentation available in the solution path. 170

THEOREM C.2. Assume notation from Theorem C.1, and that $\lambda \geq \log^{\theta_0} T$, and $N, \delta_j, j = 1, \dots, N$ lie in (c_1, c_2) for some constants $\theta_0 > 1$ and $0 < c_1 < c_2 < \infty$. Assume further that N , the number of change points, does not increase with T . Let $\hat{N}$ and $\hat{\eta}_1 < \dots < \hat{\eta}_{\hat{N}}$ be respectively the number and locations of estimated change points by seeded binary segmentation with the narrowest over threshold, the threshold of which is determined by the strengthened Schwarz information criterion using $\theta \in (1, \theta_0)$. Then there is a constant C independent of T and the decay a for seeded intervals such that, as $T \rightarrow \infty$,

$$pr \left\{ \hat{N} = N, \max_{i=1,\dots,N} \delta_i^2 |\hat{\eta}_i - \eta_i| \leq C \log T \right\} \rightarrow 1.$$

In a nutshell, seeded binary segmentation combined with the narrowest over threshold selection has the same statistical guarantee as the original narrowest over threshold method of Baranowski et al. (2019), but requires far less computation. The worst case computational complexity is nearly linear for a single threshold based selection, see Theorem 1, part 2) and the full solution path also seems to scale nearly linear empirically, see Fig. B.1. 175

Remark 1. Assume that the minimal segment length $\lambda = \min_{i=0,\dots,N} |\eta_{i+1} - \eta_i|$ is a priori known to be $\geq O(T^\gamma)$, for some $\gamma \in [0, 1)$. Note that this is always the case for $\gamma = 0$. Then it is sufficient to construct seeded intervals up to scale $O(T^\gamma)$, i.e., remove all seeded intervals that are of length $\leq O(T^\gamma)$. This modification of seeded intervals has no impact on the statistical performance, namely, the guarantees formulated in Theorems C.1 and C.2 (i.e., Theorem 2) remain valid. The worst case computational complexity is still of order $T \log T$, as in Theorem 1, but the leading constant becomes smaller, thus, leading to a faster run time in practice. In particular, for larger values of γ , there are much fewer generated intervals than the worst case $O(T)$ for $\gamma = 0$ and thus selection procedures can be significantly faster. Moreover, reducing the number of intervals allows the construction of methods with a sub-linear number of evaluations of the gain functions that is particularly appealing e.g. in certain high-dimensional problems where each evaluation involves fitting costly models. Specifically, Kovács et al. (2020b) proposed adaptive and fast optimistic search strategies, replacing the traditional full grid search that evaluates gains at all possible split points by an optimistic search that requires only a logarithmic number of evaluations of the gain function by looking at only a logarithmic number of split points. Kovács et al. (2020b) applied optimistic search within seeded intervals, calling this variant the optimistic seeded binary segmentation method, and showed very strong statistical guarantees while requiring sub-linear total number of evaluations as long as $\gamma > 0$.

D. AN EXAMPLE OF SEEDED DYNAMIC PROGRAMMING: FAST APPROXIMATIONS OF OPTIMAL PARTITIONS BY COMBINING SEEDED INTERVALS AND DYNAMIC PROGRAMMING

Consider the problem of finding the best segmentations of a time series according to a certain loss function into exactly $1, \dots, K_{max}$ pieces. In the univariate Gaussian setting with constant variance, as used throughout this paper, as an example, one would minimize the sum of squared errors. Various dynamic programming algorithms (Auger & Lawrence, 1989; Rigai, 2015) have been developed to solve this task. The well-known segment neighbourhood algorithm (Auger & Lawrence, 1989) computes a path of these $1, \dots, K_{max}$ partitions in $O(K_{max}T^2)$ time for a time series of length T . One of course needs to have prior knowledge to be sure that the true number of change points N is less than K_{max} , otherwise there is no chance of finding all change points. Assume now that an oracle says that out of the $T - 1$ possible locations, the change points can be located only at a smaller subset of $K < T - 1$ locations. Armed with such additional information, one could restrict the segment neighbourhood algorithm to work only with these candidates. The solution could be computed in $O(K_{max}K^2 + T)$ time. The second term comes from the fact that one still needs to go once over the whole time series to compute necessary costs, e.g. sum of squared errors, in segments between the K candidate change point locations. In general $N < K_{max} < K$. In the best case $K = N = K_{max}$, i.e., one is essentially told the correct number and location of change points. Then restricting the segment neighbourhood algorithm to these K candidates is much faster than searching through all $T - 1$ possible split points. For example, one could obtain a solution path with linear complexity $O(T)$ as long as the number of change points is of a smaller order than the cube root of T . Note that this does not imply anything about their minimal spacing. In the worst case $K = T - 1$, one is back to the original segment neighbourhood algorithm with $O(K_{max}T^2)$ computational cost. The latter implies, that in the presence of many change points, e.g. in frequent change point scenarios, this algorithm cannot improve much on the original computational times. However, in such frequent change point scenarios, the desire is probably not to provide a full solution path anyway.

In practice there is no oracle telling the exact change point locations, but K split points that have the highest corresponding gain values in various seeded intervals can be used to form a promising set of candidates. While having no guarantee any longer for finding exactly the optimal segmentations in the solution path, the hope is that one can still have fairly good approximations given that seeded intervals cover all change points well and those intervals that cover single change points should have high corresponding maximal gains as long as the jump is reasonably large. Note that all change points are covered by at most $O(\log T)$ seeded intervals, hence, K can be chosen as having the order of roughly $N \log T$. In practice this can be even closer to N , as often the best split point candidates in various overlapping

background intervals are identical. Given the additionally needed evaluation of the gains in seeded intervals, the overall cost of the above described algorithm would be $O(K_{max}K^2 + T + T \log T)$. We call this combination of selecting candidates from seeded intervals followed by the segment neighbourhood algorithm restricted to this selected set of candidates as the seeded segment neighbourhood algorithm. In principal, other dynamic programming algorithms (see e.g. Killick et al. (2012); Rigaiil (2015); Maidstone et al. (2017) and references therein) could be explored too, thus forming the set of seeded dynamic programming algorithms. For example, rather than finding a solution path, one could also consider the problem of penalized costs. Note that Rigaiil (2015) described a dynamic programming algorithm based on functional pruning that also significantly improves computational times compared to segment neighbourhood in practice. However, that algorithm currently works only for a unidimensional parameter and hence its applicability is restricted. As a comparison, the seeded segment neighbourhood algorithm would be more flexible, it merely needs a different gain function for finding the candidates in seeded intervals.

Table 1: Simulation results for best segmentations and their approximations

signal	T	N / K	meas	best partitioning found with the following number of change points							
				4	5	6	7	9	10	11	12
blocks	2048	11 / 33	avHD	0.13	0.68	1.16	0.68	1.50	2.38	14.61*	61.99
blocks	2048	11 / 33	%HD	96	90	92	86	84	82	69 *	26
blocks	2048	11 / 55	avHD	0.13	0.55	1.16	0.60	1.04	1.12	5.89 *	35.23
blocks	2048	11 / 55	%HD	96	91	92	88	86	86	80 *	52
fms	497	6 / 18	avHD	0.38	0.74	0.69 *	23.14	45.40	53.23	58.17	62.83
fms	497	6 / 18	%HD	94	90	88 *	48	17	8	2	1
fms	497	6 / 30	avHD	0.15	0.43	0.39 *	13.12	18.32	25.05	28.57	39.35
fms	497	6 / 30	%HD	96	93	92 *	70	48	42	28	19

T, number of observations for the signal; N, number of true change points; K, number of top candidates taken from seeded intervals; meas, measure; avHD, average Hausdorff distance over 100 simulations between the best segmentation found by segment neighbourhood for all possible split points and the approximation thereof using seeded segment neighbourhood with K candidates and decay $a = 1/2^{1/2}$; %HD, percentage of simulations where HD is exactly equal to 0, i.e., the percentage of cases where the approximation found the truly best segmentation. The star symbol marks the segmentation according to the true underlying number of change points for the corresponding signal. Signals are described in Appendix B of Fryzlewicz (2014), but note that the name for the fms signal is misleading as this signal first appeared in Olshen et al. (2004).

Table 1 shows how well the seeded segment neighbourhood algorithm can approximate the best segmentations compared to the original segment neighbourhood algorithm that is guaranteed to find the optimal segmentations with the specified number of change points. We set the number of top candidates taken from seeded intervals as three and five times larger than the true number of change points for the above explained reason that each change point is contained in multiple seeded intervals and these might lead to slightly different candidates. We thus take these slightly bigger pools of candidates. For segmentations into fewer segments than the true underlying number, the average Hausdorff distance between the truly optimal segmentation and our approximation thereof using seeded segment neighbourhood is very low. For example, the average Hausdorff distance is 5.89 observation for the blocks signal with 11 change points and $K = 55$ candidates from seeded intervals. This should be compared to the total of 2048 observations. Also shown is the percentage of simulations where the approximation actually found exactly the same segmentation as the truly optimal one. For the fms signal, when considering segmentations with the true underlying number of 6 change points, the exact same segmentation is obtained in 88 percent of the simulations. Segmenting into a larger number of segments, the Hausdorff distances increase. This is due to the fact that once the true change points have been identified, the additionally added ones are only due to randomness and hence have a somewhat random location. As their gain may not be very high, they might not make it onto the shortlist of top K candidates from seeded intervals. Such larger approximation errors in the solution path beyond the true number of change points are not really a concern in practice, because

one would not select such a segmentation anyway. Overall, it seems like in these two examples the seeded segment neighbourhood algorithm does a fairly good job of approximating the best segmentations.

E. SIMULATIONS FOR THE UNIVARIATE GAUSSIAN CASE

Examples 1 and 2 in the Introduction illustrated (see Fig. 1 in the main paper) that in longer signals with short minimal spacing between consecutive change points the estimation performance of seeded binary segmentation variants are all much better than wild binary segmentation or the narrowest over threshold method, unless drawing a huge number of random intervals. To see that this does not only occur in “extreme scenarios” such as those two examples, we consider another example. Figure E.1 illustrates the statistical efficiency vs. computational efficiency of seeded binary segmentation compared to wild binary segmentation for a univariate signal with 55 change points. The total length of the search intervals (and thus the computational cost) is orders of magnitude smaller for the same statistical performance. Moreover, the performance of the default option ($a = 1/2^{1/2}$) for seeded binary segmentation is very close to what seems to be possible in this specific setup. Choosing a slower decay slightly improves estimation, but the results are fairly robust across the number of generated seeded intervals (unlike random intervals, where too few of them leads to a clearly worse estimation performance).

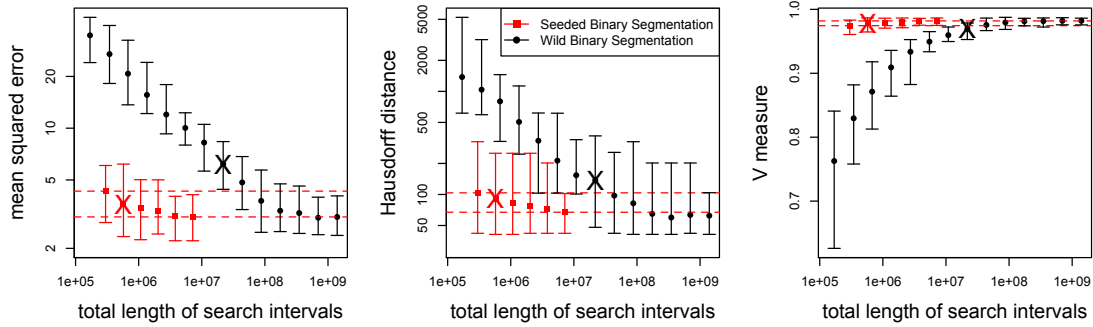


Fig. E.1: Estimation performance on a five times repeated blocks signal (Donoho & Johnstone, 1994) with $5 \cdot 2048$ observations and 55 change points. The dots and squares represent the average and the error bars the range of obtained values across 50 simulations. Crosses correspond to the proposed default options for the methods. The dashed horizontal lines are drawn as a visual aid. Seeded binary segmentation was combined with greedy selection and the final models were chosen using the same information criterion as for wild binary segmentation. Note the logarithmic scales on both axes.

We consider additionally the same simulations as presented by Fryzlewicz (2014) in connection with wild binary segmentation. For a description of the signals see Appendix B of Fryzlewicz (2014). We used implementations of the wild binary segmentation and the narrowest over threshold method from the R packages **wbs** and **not** from CRAN, respectively, with a small bug fixed in both packages concerning the generation of random intervals. A fast Fortran based implementation of our proposed seeded binary segmentation methodology (currently with GT, GS and NT selection) is available at <https://github.com/kovacssolt/ChangePoints>. Arbitrary intervals as inputs were already supported by the **not** package, and this package allows using our method for more general signals, e.g. with changing slopes. Note however, that the latter implementation can be slow as all segmentations in the solution path resulting from all possible thresholds are recorded, which leads to a quadratic number of candidates being saved.

Table 2 shows results based on averaged values over 100 simulations for each setup. Reported are the mean squared errors (MSE), Hausdorff distances, V measures (Rosenberg & Hirschberg, 2007, more precisely, viewing the change point segmentation as a clustering problem by identifying each constant

Table 2: Simulation results

signal	method	MSE	Hausdorff dist.	V measure	$\hat{N} - N$	tot. length
blocks	SeedBS-GS	2.922 (1.077)	43.150 (31.009)	0.970 (0.013)	-0.610 (0.803)	95.3 (0)
	SeedBS-GS*	2.685 (0.828)	32.690 (21.037)	0.973 (0.009)	-0.520 (0.577)	329.7 (0)
	WBS	2.627 (0.854)	31.520 (23.262)	0.973 (0.011)	-0.500 (0.577)	3419.1 (35.7)
	SeedBS-NS	2.942 (1.002)	42.630 (28.690)	0.970 (0.013)	-0.690 (0.787)	95.3 (0)
	NOT	2.539 (0.874)	31.700 (22.778)	0.974 (0.011)	-0.590 (0.588)	3422.3 (32.6)
	SeedBS-NS*	2.675 (0.782)	33.790 (21.056)	0.973 (0.010)	-0.600 (0.532)	329.7 (0)
fms	SeedBS-GS	0.005 (0.004)	15.810 (25.830)	0.955 (0.037)	-0.020 (0.492)	19.1 (0)
	WBS	0.004 (0.003)	9.080 (14.975)	0.963 (0.031)	0.000 (0.284)	833.8 (7.9)
	SeedBS-NS	0.004 (0.003)	15.500 (25.853)	0.958 (0.035)	-0.040 (0.448)	19.1 (0)
	NOT	0.004 (0.002)	8.640 (15.019)	0.967 (0.028)	-0.010 (0.266)	834.4 (7.4)
mix	SeedBS-GS	1.598 (0.517)	86.870 (57.740)	0.908 (0.044)	-1.180 (1.048)	22.3 (0)
	WBS	1.680 (0.728)	78.260 (58.431)	0.913 (0.041)	-1.090 (1.093)	939.1 (8.1)
	SeedBS-NS	1.759 (0.605)	96.870 (64.631)	0.897 (0.052)	-1.340 (1.199)	22.3 (0)
	NOT	1.981 (0.914)	89.220 (64.899)	0.903 (0.049)	-1.260 (1.495)	939.2 (8.3)
teeth10	SeedBS-GS	0.061 (0.040)	7.960 (20.229)	0.933 (0.130)	-0.190 (2.419)	4.4 (0)
	WBS	0.058 (0.040)	7.380 (20.591)	0.936 (0.133)	-0.410 (2.170)	238.7 (2.2)
	SeedBS-NS	0.066 (0.050)	10.790 (27.521)	0.911 (0.186)	-0.860 (2.903)	4.4 (0)
	NOT	0.062 (0.044)	9.150 (24.593)	0.923 (0.162)	-0.730 (2.522)	238.8 (2.3)
stairs10	SeedBS-GS	0.023 (0.011)	2.130 (1.468)	0.981 (0.013)	0.470 (0.745)	4.8 (0)
	WBS	0.024 (0.013)	1.900 (1.374)	0.980 (0.014)	0.450 (0.730)	255.4 (2.3)
	SeedBS-NS	0.021 (0.011)	1.340 (1.249)	0.984 (0.013)	0.100 (0.362)	4.8 (0)
	NOT	0.021 (0.011)	1.400 (1.326)	0.984 (0.013)	0.140 (0.450)	255.5 (2.5)

SeedBS-GS, Seeded binary segmentation with greedy solution path; SeedBS-NS, Seeded binary segmentation with narrowest over threshold solution path; WBS, Wild binary segmentation; NOT, Narrowest over threshold method; MSE, Mean squared error; Hausdorff dist., Hausdorff distance; tot. length, total length of search intervals in thousand. Signals are described in Appendix B of Fryzlewicz (2014), but note that the name for the fms signal is misleading as this signal first appeared in Olshen et al. (2004). Seeded binary segmentation variants use the decay $a = 1/2^{1/2}$ with the exception of the cases marked with a star, where $a = 1/2^{1/8}$ was used

part as a cluster), errors in the estimates for the number of change points as well as the total length of the constructed search intervals. Parentheses show the corresponding standard deviations. For our seeded binary segmentation methodology we chose $m = 2$ and $a = 1/2^{1/2}$. Exceptions are the rows marked with a star, where the decay was chosen to be $a = 1/2^{1/8}$, i.e. an increase in the total length of search intervals by a factor of roughly four. For both wild binary segmentation and the narrowest over threshold methods we chose the recommended 5000 random intervals. Moreover, all methods selected the final model from the solution path using the same information criterion as in equation (4) of Fryzlewicz (2014). In detail, the variances are assumed to be unknown and one chooses the segmentation k in the solution path that optimizes $T/2 \log(\hat{\sigma}_k^2) + k \log T$. For more details on this criterion, see its first appearance in Yao (1988).

Overall, seeded binary segmentation with a greedy solution path (SeedBS-GS) performs similarly to wild binary segmentation (WBS), while seeded binary segmentation with the narrowest over threshold solution path (SeedBS-NS) is similar to the original narrowest over threshold method (NOT). Most importantly, using seeded intervals we achieve competitive performance with total lengths of search intervals that are orders of magnitudes smaller than using the random intervals, and thus computational times are massively reduced. In these scenarios both selection types perform similarly. We have the experience that increasing the noise level would lead to better results using the greedy selection. The performance of our default methodology (with $a = 1/2^{1/2}$) is slightly worse for the blocks signal compared to random intervals based counterparts. However, when choosing $a = 1/2^{1/8}$ (see rows marked with stars), and thus accepting an increase in the total length of the search intervals by about a factor four, performances are again very close. Note that the total length of search intervals is still only a tenth of that of random in-

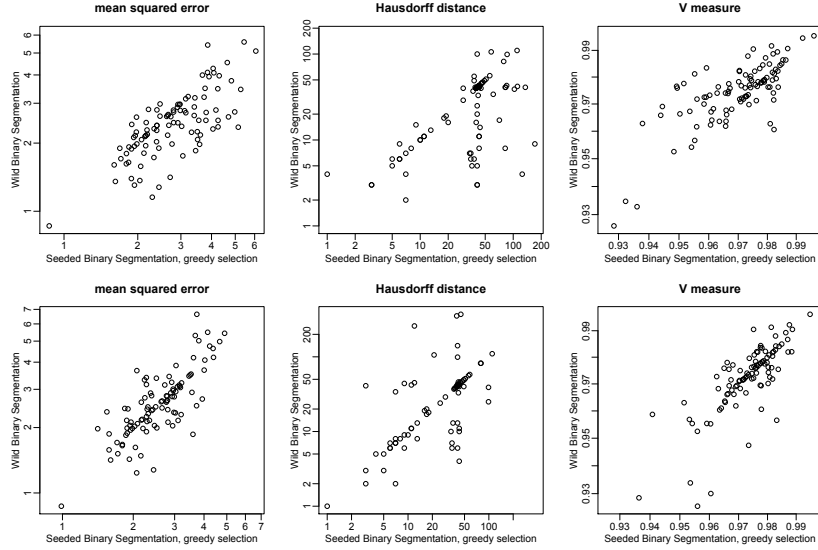


Fig. E.2: Simulation results from Table 2 for the blocks signal with decay $a = 1/2^{1/2}$ (top) and decay $a = 1/2^{1/8}$ (bottom).

tervals. For a better understanding, the simulations for the blocks signal are depicted in Fig. E.2. In the top row (with $a = 1/2^{1/2}$) one can see points lining up vertically when considering Hausdorff distances. This line is around 40, and indicates that one of the true change points (lying to this distance to its closest neighbour) is missed. However, when increasing computational effort ($a = 1/2^{1/8}$, bottom row), this vertical line basically disappears and for all three error measures many points line up exactly on the identity line. This indicates that change points found using seeded binary segmentation with greedy selection and wild binary segmentation exactly agree in many simulation runs.

E.1. Improvements of estimation

Besides increasing the computational effort by the choice of a slower decay a (see different values of α for the blocks signal in Table 2), there are many other options to slightly improve empirical performance. For example, one could find preliminary change point candidates with the recommended choice of $a = 1/2^{1/2}$ and include additional search intervals in a second step covering only the ranges in between previously found neighbouring change points. Should the best candidates from these additional intervals have test statistics above the predefined acceptance threshold, then one can also add them to the final set of change point estimates. Implementing to add these intervals is slightly more effort than just changing the decay a , but in simulations it seemed to be a bit more efficient computationally and in terms of estimation performance than just the brute force choice of a slower decay value. The reason is that the additional intervals are added adaptively, i.e. taking into consideration already detected change points.

One could of course also use the seeded dynamic programming approaches described in Section D as a kind of “post-processing”, i.e. use seeded binary segmentation to obtain initial candidates, and then let the dynamic programming do a refinement on this restricted set of candidates. Another option mentioned in the definition of seeded intervals is to limit the minimal number of observations m (i.e. consider fewer layers) if prior information on the minimal true segment lengths is available. This reduces the risk of selecting false positives coming from very small intervals during the final model selection. One could also use more refined test statistics instead of the CUSUM statistics, ones that are adapted to various scales, e.g. multiscale variants. Simulations for these above mentioned improvement options showed estimation performances that are very close to the best competitors, and yet retain computational efficiency. However, we omit the corresponding detailed simulation results due to spatial reasons.

F. DETAILS OF HIGH-DIMENSIONAL MODELS

Another appealing applicability of seeded binary segmentation beyond the extensively discussed univariate Gaussian setup is in scenarios where model fits are expensive, for example in the recently emerging area of high-dimensional change point detection (e.g. means in Soh & Chandrasekaran (2017); Wang & Samworth (2018); covariance and precision matrices/graphical models in Roy et al. (2017); Gibberd & Nelson (2017); Wang et al. (2021); Gibberd & Roy (2017); Bybee & Atchadé (2018); Avanesov & Buzun (2018); Lonschien et al. (2021); linear regression in Leonardi & Bühlmann (2016); Wang et al. (2021)). Several of these proposals require the calculation of numerous lasso (Tibshirani, 1996), graphical lasso (Friedman et al., 2008) or similar rather costly estimators. Another example would be multivariate non-parametric change point detection for which Lonschien et al. (2022) use random forests (Breiman, 2001) and other classifiers, Padilla et al. (2022) use kernel density estimation, while Matteson & James (2014) utilise energy distances. Overall, in all approaches requiring costly single fits, it is essential to keep the total length of search intervals as small as possible to make computations feasible and thus to have practically usable methods.

We would like to highlight this in a specific example involving change point detection in (high-dimensional) Gaussian graphical models. We consider a setup and estimator as introduced by Lonschien et al. (2021). Let $(X_i)_{i=1}^T \subseteq \mathbb{R}^p$ be a sequence of independent Gaussian random vectors with means $\mu_i \in \mathbb{R}^p$ and covariance matrices $\Sigma_i \in \mathbb{R}^{p \times p}$ such that $(\mu_i, \Sigma_i), i = 1, \dots, T$ is piecewise constant with an unknown number N of change points at unknown locations and corresponding segment boundaries $1 = \eta_0 < \eta_1 < \dots < \eta_N < \eta_{N+1} = T$. For $0 \leq u < v \leq T$ let $X_{(u,v]} \in \mathbb{R}^{p \times (v-u)}$ denote the matrix of the observations $X_{u+1}, \dots, X_v$, denote with $\hat{\mu}^{(u,v]}$ their mean and let $S_{(u,v]} = (X_{(u,v]} - \hat{\mu}^{(u,v]}) (X_{(u,v]} - \hat{\mu}^{(u,v]})^t / (v - u)$ be the corresponding covariance matrix. Let $0 < \delta < 1/2$ be some required minimal relative segment length and $\lambda > 0$ be a regularisation parameter. According to the proposal of Lonschien et al. (2021), for a segment $(u, v]$ with $v - u > 2\delta T$ define the split point candidate as

$$\hat{\eta}_{(u,v]} = \arg \max_{s \in \{u+\delta T, \dots, v-\delta T\}} L_T(\hat{\Omega}_{(u,v]}^{\text{glasso}}; S_{(u,v]}) - \left(L_T(\hat{\Omega}_{(u,s]}^{\text{glasso}}; S_{(u,s]}) + L_T(\hat{\Omega}_{(s,v]}^{\text{glasso}}; S_{(s,v]}) \right), \quad (1)$$

where L_T corresponds to a multivariate Gaussian log-likelihood based loss in the considered segment, i.e.

$$L_T(\Omega; S_{(u,v]}) = \frac{v-u}{T} \left(\text{Tr}(\Omega S_{(u,v]}) - \log(|\Omega|) \right),$$

for an arbitrary segment $(u, v]$. Moreover, $\hat{\Omega}_{(u,v]}^{\text{glasso}}$ is the graphical lasso precision matrix estimator (Friedman et al., 2008) for the corresponding segment with a specifically scaled regularisation parameter, i.e.

$$\hat{\Omega}_{(u,v]}^{\text{glasso}} = \arg \min_{\mathbb{R}^{p \times p} \ni \Omega \succ 0} L_T(\Omega; S_{(u,v]}) + \sqrt{(v-u)/T} \lambda \|\Omega\|_1.$$

The goal here is to compare the estimation performance and computational times of the estimator (1) when applied to seeded as well as random intervals. We consider a specific chain network model (Fan et al., 2009, Example 4.1) with $p = 20$ variables: $\Sigma_{ij} = \exp(-a |s_i - s_j|)$ with $a = 1/2$ and $s_i - s_{i-1} = 0.75, i = 2, \dots, p$. The resulting precision matrix Σ^{-1} is tridiagonal. We consider a modified version $\tilde{\Sigma}$ by replacing the top left 5×5 part of Σ by I_5 , i.e. an identity matrix of dimension 5. Let $F_1 = \mathcal{N}(0_p, \Sigma)$ and $F_2 = \mathcal{N}(0_p, \tilde{\Sigma})$. We take 20 segments of length 100 each (i.e., $N = 19$) and generate independent observations in the following way. The first hundred observations are drawn from F_1 , the following hundred from F_2 , then another hundred from F_1, F_2, F_1 etc. In total, we obtain $T = 2000$ observations. We simulated 10 different data sets this way and to each of these applied seeded binary segmentation (greedy selection with a single fixed threshold) with decays $a = 1/2, 1/2^{1/2}, 1/2^{1/4}, 1/2^{1/8}, 1/2^{1/16}$ as well as $M = 100, 200, 400, 800, 1600$ random intervals for wild binary segmentation, leading to the 5 red, respectively 5 black point clouds in Fig. F.1 (which is a zoomed in version of Fig. 3 in the main paper). To be precise, we set $\delta = 0.015$, hence, we left out 30 observations at the boundaries of background intervals. This also implies that all intervals with fewer than 60 observations were discarded for both interval construction types. We set $\lambda = 0.007$ and a threshold of 0.04 as a necessary improvement in (1) in order to keep the proposed split point $\hat{\eta}_{(u,v]}$ in the segment $(u, v]$ and declare it as a change point.

Note that with this specific choice of δ we do not have a truly high-dimensional scenario in any of the segments, but regularisation is still helpful especially when considering short background intervals where the number of variables p is of similar magnitude as the number of observations. The reasons why we kept the number of variables p comparably low are on the one hand the easier choice of tuning parameters, and on the other hand we would quickly face computational bottlenecks with wild binary segmentation when increasing p . In this 20-dimensional example, around 2300 seconds is needed for wild binary segmentation to obtain an estimation accuracy that is already achieved in about 28 seconds using seeded intervals, see top of Fig. F.1. The bottom of Fig. F.1 shows computational gains of similar magnitude (17 vs. 1200 seconds). The difference in computational times for the top and bottom of Fig. F.1 arises from the different implementation. The top shows a naive implementation where for each split point $s \in (u, v]$ both the input covariance matrices ($S_{(u,s]}$, $S_{(s,v]}$) as well as the corresponding graphical lasso fits ($\hat{\Omega}_{(u,s]}^{glasso}$, $\hat{\Omega}_{(s,v]}^{glasso}$) are computed from scratch. However, the input covariance matrices can be updated cheaply for neighbouring split points, i.e. once $S_{(u,s]}$ is available, with a cost of $O(p^2)$ it can be updated to obtain $S_{(u,s+1]}$. This better implementation, shown at the bottom part of Fig. F.1, saves 40 – 50% in computational time both for wild and seeded binary segmentation, but still preserves the massive computational gains of the seeded intervals. The remaining computational costs mainly come from the expensive graphical lasso fits. Without computing the graphical lasso fits, in our R implementation one would only need about 2 seconds for the fastest version of seeded binary segmentation ($a = 1/2$) and about 9 seconds for $M = 100$ and 150 seconds for $M = 1600$ for wild binary segmentation. However, as mentioned previously, without the regularised fits of the graphical lasso, the estimation error would be clearly worse, and hence, this is not an option. Thus, the next best guess to speed up would be to use warm starts from neighbouring splits when updating the graphical lasso fits, similar to the covariance matrix updates described previously. Unfortunately, not all algorithms that have been developed to compute graphical lasso fits are guaranteed to converge with warm starts. Mazumder & Hastie (2012b) proposed the DP-GLASSO algorithm that can be started at any positive definite matrix. When using the corresponding **dpglasso** R package on CRAN, we obtained speedups of 30 – 35% with warm starts, but even this way the computational times were roughly 10 – 20 times worse than using the graphical lasso algorithm of Friedman et al. (2008) implemented in the **glasso** R package on CRAN. The latter one we used to obtain the results of Fig. F.1. Admittedly, there are a lot of factors influencing computational times (e.g. convergence thresholds) and different algorithms/implementations might be better or worse in other scenarios with a higher number of variables p and/or regularisation parameter λ . One could of course also try other precision matrix estimators, e.g. the graphical elastic net (Kovács et al., 2021) with its fast implementation in the **GLassoElnetFast** package; or others based on adaptive lasso or scad penalties (Fan et al., 2009). However, the computational times would be of similar magnitude.

It is hard to draw an overall conclusion on which combination of implementation and tricks would give the fastest results for changing Gaussian graphical models, but it is even harder to be fast without utilising the more efficient seeded interval construction for the following reason. In each background interval evaluating (1) at a single split point s requires the calculation of 2 graphical lasso fits. Hence, the total computational cost (for a small δ) would be roughly $O(T \log(T)p^2 + T \log(T) \cdot \text{gl}(p))$ for seeded binary segmentation versus $O(TMp^2 + TM \cdot \text{gl}(p))$ for wild binary segmentation, where $\text{gl}(p)$ denotes the cost of computing a graphical lasso estimator for a p -dimensional covariance matrix. There are various algorithms computing exact or approximate solutions for the graphical lasso estimator with differing computational cost, but scalings for $\text{gl}(p)$ of $O(p^3)$ (or even worse) are common once the input covariance matrices are already available and unless some special structures can be exploited (e.g. block diagonal screening proposed by Witten et al., 2011 as well as Mazumder & Hastie, 2012a). The terms $T \log(T)p^2$ and TMp^2 come from the calculation of the input covariance matrices via updates. Summarising all observations from above, for typical algorithms for graphical lasso problems, the total costs would be roughly $O(T \log(T)p^3)$ for seeded binary segmentation vs. $O(TMp^3)$ for wild binary segmentation.

As in the univariate Gaussian change in mean case, M needs to be much larger than $O(\log T)$ if there are many change points. In the worst case M is of the order $O(T^2)$ when evaluating essentially all possible background intervals. In practice, one often does not know which choice of M is sufficient. This is a

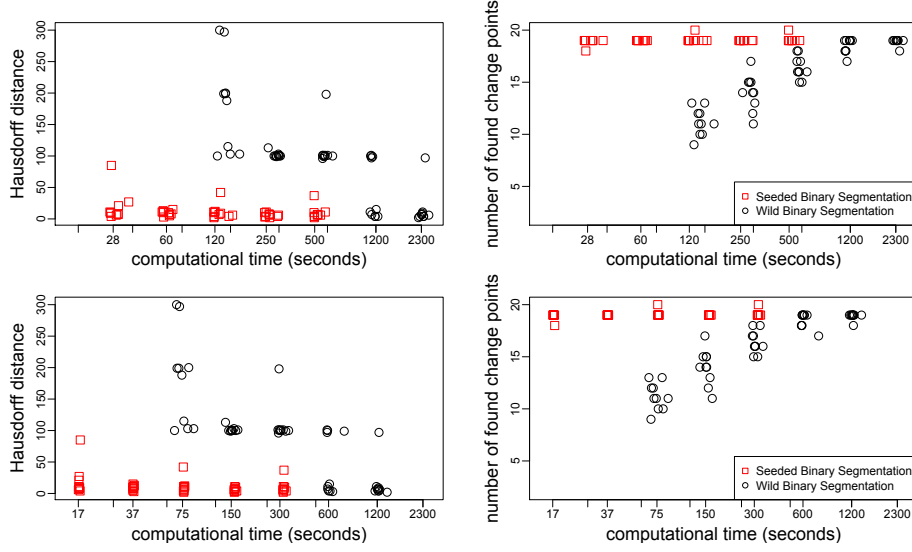


Fig. F.1: This figure is a zoomed in version of Fig. 3 in the main paper, showing estimation performance in the changing Gaussian graphical model example of Section F with 2000 observations and 19 change points with the naive implementation (top) and the implementation using updates for neighbouring input covariance matrices (bottom). Note the logarithmic scales.

problem, as the performance very much depends on the choice of M (see Fig. F.1), especially when there are many change points. Another advantage of seeded intervals is, that results are not depending much on the choice of the decay a for the seeded interval construction compared to the very strong dependence on the choice of M (the number of random intervals) for wild binary segmentation.

Overall the aim of this example is to demonstrate that seeded binary segmentation is highly useful in scenarios beyond the univariate Gaussian change in mean case. While we precisely quantify the computational gains in the changing Gaussian graphical model scenario when using graphical lasso estimators, it is more difficult to quantify this for some other costly estimators such as the lasso (Tibshirani, 1996) or random forest (Breiman, 2001) for the following reasons. The main difference is that they have a cost that not only depends on the number of variables, but also on the number of observations, moreover, possibilities to update neighbouring fits would need to be considered carefully. Even in absence of exact quantification of the computational costs in these scenarios, one can expect the substantial computational gains when using seeded intervals to be crucial to keep computations feasible when model fits are very expensive.

It is worth noting that recently Kovács et al. (2020b) proposed to replace the full grid search in such costly high-dimensional scenarios with an adaptive optimistic search technique requiring many fewer fits. Specifically, the traditional full grid search requires $O(T)$ costly model fits for an interval of length T , whereas optimistic search looks at only $O(\log T)$ split points and thus requires only a logarithmic number of costly fits. Hence, using this optimistic search technique in seeded intervals can lead to further significant speedups and thus seems to be a promising avenue to tackle computational challenges in such high-dimensional change point detection problems.

G. PROOFS

Proof of Theorem 1, parts 1) and 2). The algorithm consists of two steps: the calculation of the CUSUM statistics for all seeded intervals and the subsequent model selection. For the first step, the cost of calculating CUSUM statistics on all seeded intervals is proportional to their total length, i.e., $O(T \log T)$.

In the second step, we only consider the candidate split points from seeded intervals with maximal CUSUM values above the given threshold, and sort them either according to their maximal gains (for the greedy selection) or their lengths (for the narrowest over threshold selection). Note that the number of intervals contained in a collection of seeded intervals is of the order $O(T)$. Hence, the sorting procedure involves $O(T \log T)$ computations in the worst case. Then we check for each seeded interval in the sorted list whether it contains any previously found change point. Such a check can be done in $O(\log T)$ computations as follows. We store all previously found change points in a balanced binary tree structure (e.g., the AVL tree by Adel'son-Vel'skiĭ & Landis, 1962). An alternative tree structure that can be used easily instead of balanced binary search trees is shown in Fig. G.1. For AVL trees, both the search and the insertion of new elements have the worst case complexity $O(\log T)$, and the memory complexity is $O(T)$ (given the maximum of T total points that could potentially be declared as change points and thus need to be stored in the AVL tree). Thus, one needs $O(\log T)$ time to check whether the candidate split is suitable (i.e. the corresponding search interval does not contain any previously found change point, for which finding the nearest neighbouring points in the AVL tree to the currently considered candidate is sufficient), and $O(\log T)$ time for updating the AVL tree supposing that the candidate is suitable and thus that it is declared/detected as a new change point. Alternatively, it is possible to utilise the structure of seeded intervals by noticing that each potential split point $s \in \{1, \dots, T\}$ is contained in at most $O(\log T)$ intervals (as in each of the $O(\log T)$ scales there is a constant number of intervals containing s). This allows the marking of all of the intervals that contain the found change point with a cost of $O(\log T)$ such that these intervals can then be skipped in consecutive steps when adding the new change points. Overall, both approaches (AVL trees and the specific structure of seeded intervals) lead to a worst case complexity $O(T \log T)$ for the second step as we go through at most $O(T)$ intervals (the total number of intervals in a collection of seeded intervals) and perform at most $O(\log T)$ computations at each (checking whether the interval is suitable and an update of the AVL tree). Therefore, the total computational complexity is $O(T \log T)$, and it does not depend on the underlying number of true change points. The total memory complexity is composed of the storage of information (maximal gain, best split, as well as start and end points for each search interval) plus the AVL tree, all of which requires at most $O(T)$ memory. $\square$

Proof of Theorem 1, part 3). In order to apply the information criterion, we need to compute the full solution path, namely, the solutions for all possible choices of thresholds. Consider first the greedy selection, which has the nice property that the obtained solutions are nested, i.e. adding another split point (equivalent to lowering the threshold) splits one of the existing intervals into two parts. Hence, to obtain the full solution path, one can just let the model selection run as in Theorem 1, part 1), but with a threshold set to 0, leading to at most $O(T \log T)$ computations. Further, the penalty $\text{PEN}(\cdot)$ can be evaluated in an incremental way, i.e., each time when adding a new change point, which by assumption needs $O(\log T)$ computations per update. For the least squares part in the information criterion, it is also possible to update incrementally with $O(\log T)$ computations at each step. More precisely, we have to find the nearest left and right points (out of all change points previously found) to the newly added change point. Once the neighbours are found, one can just update the least squares of the specific segment that was freshly divided in $O(1)$ time by utilising cumulative sum type of information (all of which can be precomputed once in the beginning in $O(T)$ time, and stored in $O(T)$ space). Such searches of nearest neighbours can be done in $O(\log T)$ time, if we build and iteratively update a balanced binary tree (e.g., the AVL tree by Adel'son-Vel'skiĭ & Landis, 1962) for the change points found, as in the proof of part 1) and 2). Note that such balanced trees can be updated incrementally also in $O(\log T)$ time. Given that we go through at most $O(T)$ intervals and perform at most $O(\log T)$ computations at each, an upper bound on the worst case computational complexity of $O(T \log T)$ for the greedy selection is obtained. $\square$

Note that regarding the narrowest over threshold selection with full solution path (NS) we could do the following (potentially suboptimal analysis). There are at most $O(T)$ seeded search intervals and thus thresholds which can potentially lead to different solutions. For each threshold, we can compute the solution in $O(T \log T)$ time by Theorem 1, part 2) and evaluate the information criterion in $O(T \log T)$ time (i.e., in an incremental way similarly to the greedy selection considered above). Thus, in total, an upper

505

510

520



Fig. G.1: Example of an interval tree structure indicating the number of change point candidates contained in specific segments. At the lowest level, change points are marked with ones, the others with zeros. The nodes in the levels above are the sum of node values from the previous layer and thus count the number of change point candidates within specific intervals. The candidates in this example are at observations 3, 5, 6 and 13. Thus, the node on level 4 indicates that there are 4 change points overall in the interval $\{1, \dots, 16\}$. The left and right nodes on level 3 indicate that there are 3 and 1 candidates in the intervals $\{1, \dots, 8\}$ and $\{9, \dots, 16\}$, respectively. The leftmost node on level 2 indicates that there is only one candidate in the interval $\{1, \dots, 4\}$. The remaining nodes have analogous interpretations.

Proof of Theorem C.1. We consider interval $(\eta_i - \lambda, \eta_i + \lambda]$ for each change point η_i . By construction of seeded intervals, we can find a seeded interval $(c_i - r_i, c_i + r_i]$ such that $(c_i - r_i, c_i + r_i] \subseteq (\eta_i - \lambda, \eta_i + \lambda]$, $r_i \geq a^2 \lambda$ and $|c_i - \eta_i| \leq r_i(1 + a^2)/2$. Then the problem almost reduces to the detection of change point on the interval $(c_i - r_i, c_i + r_i]$. The rest of the proof follows nearly the same as the proof of Theorem 1 in Baranowski et al. (2019), and is thus omitted. $\square$

Proof of Theorem C.2. It follows exactly the same way as the proof of Theorem 3 in Baranowski et al. (2019). $\square$

REFERENCES

- ADEL'SON-VEL'SKIĬ, G. M. & LANDIS, E. M. (1962). An algorithm for organization of information. *Doklady Akademii Nauk SSSR* **146**, 263–266.
- ANASTASIOU, A. & FRYZLEWICZ, P. (2022). Detecting multiple generalized change-points by isolating single ones. *Metrika* **85**, 141–174.
- AUGER, I. E. & LAWRENCE, C. E. (1989). Algorithms for the optimal identification of segment neighborhoods. *Bulletin of Mathematical Biology* **51**, 39–54.
- AVANESOV, V. & BUZUN, N. (2018). Change-point detection in high-dimensional covariance structure. *Electron. J. Stat.* **12**, 3254–3294.
- BARANOWSKI, R., CHEN, Y. & FRYZLEWICZ, P. (2019). Narrowest-over-threshold detection of multiple change points and change-point-like features. *J. R. Statist. Soc. B* **81**, 649–672.
- BOYSEN, L., KEMPE, A., LIEBSCHER, V., MUNK, A. & WITTICH, O. (2009). Consistencies and rates of convergence of jump-penalized least squares estimators. *Ann. Statist.* **37**, 157–183.
- BREIMAN, L. (2001). Random Forests. *Mach. Learn.* **45**, 5–32.
- BYBEE, L. & ATCHADÉ, Y. (2018). Change-point computation for large graphical models: a scalable algorithm for Gaussian graphical models with change-points. *J. Mach. Learn. Res.* **19**, 440–477.
- CHAN, H. P. & CHEN, H. (2017). Multi-sequence segmentation via score and higher-criticism tests. *arXiv:1706.07586*.
- CHAN, H. P. & WALTHER, G. (2013). Detection with the scan and the average likelihood ratio. *Statist. Sinica* **23**, 409–428.
- DONOHU, D. L. & JOHNSTONE, I. M. (1994). Ideal spatial adaptation by wavelet shrinkage. *Biometrika* **81**, 425–455.
- DU, C., KAO, C.-L. M. & KOU, S. C. (2016). Stepwise signal extraction via marginal likelihood. *J. Amer. Statist. Assoc.* **111**, 314–330.
- FAN, J., FENG, Y. & WU, Y. (2009). Network exploration via the adaptive Lasso and SCAD penalties. *Ann. Appl. Stat.* **3**, 521–541.
- FRICK, K., MUNK, A. & SIELING, H. (2014). Multiscale change point inference. *J. R. Statist. Soc. B* **76**, 495–580.
- FRIEDMAN, J., HASTIE, T. & TIBSHIRANI, R. (2008). Sparse inverse covariance estimation with the graphical Lasso. *Biostatistics* **9**, 432–441.
- FRYZLEWICZ, P. (2014). Wild binary segmentation for multiple change-point detection. *Ann. Statist.* **42**, 2243–2281.
- FRYZLEWICZ, P. (2020a). Detecting possibly frequent change-points: Wild binary segmentation 2 and steepest-drop model selection. *J. Korean Statist. Soc.* **49**, 1027–1070.
- FRYZLEWICZ, P. (2020b). Detecting possibly frequent change-points: Wild Binary Segmentation 2 and steepest-drop model selection—rejoinder. *J. Korean Statist. Soc.* **49**, 1099–1105.
- GARREAU, D. & ARLOT, S. (2018). Consistent change-point detection with kernels. *Electron. J. Stat.* **12**, 4440–4486.
- GIBBERD, A. J. & NELSON, J. D. (2017). Regularized estimation of piecewise constant Gaussian graphical models: the group-fused graphical Lasso. *J. Comput. Graph. Statist.* **26**, 623–634.
- GIBBERD, A. J. & ROY, S. (2017). Multiple changepoint estimation in high-dimensional Gaussian graphical models. *arXiv:1712.05786*.
- KILLICK, R., FEARNEHEAD, P. & ECKLEY, I. A. (2012). Optimal detection of changepoints with a linear computational cost. *J. Amer. Statist. Assoc.* **107**, 1590–1598.
- KOVÁCS, S., LI, H. & BÜHLMANN, P. (2020a). Seeded intervals and noise level estimation in change point detection: a discussion of Fryzlewicz (2020). *J. Korean Statist. Soc.* **49**, 1081–1089.
- KOVÁCS, S., LI, H., HAUBNER, L., MUNK, A. & BÜHLMANN, P. (2020b). Optimistic search strategy: change point detection for large-scale data via adaptive logarithmic queries. *arXiv:2010.10194*.
- KOVÁCS, S., RUCKSTUHL, T., OBRIST, H. & BÜHLMANN, P. (2021). Graphical elastic net and target matrices: Fast algorithms and software for sparse precision matrix estimation. *arXiv:2101.02148*.
- LEBARBIER, É. (2005). Detecting multiple change-points in the mean of Gaussian process by model selection. *Signal processing* **85**, 717–736.

- LEONARDI, F. & BÜHLMANN, P. (2016). Computationally efficient change point detection for high-dimensional regression. *arXiv:1601.03704*.
- LI, H., GUO, Q. & MUNK, A. (2019). Multiscale change-point segmentation: beyond step functions. *Electron. J. Stat.* **13**, 3254–3296.
- LONDSCHIEN, M., BÜHLMANN, P. & KOVÁCS, S. (2022). Random Forests for change point detection. *arXiv:2205.04997*. 590
- LONDSCHIEN, M., KOVÁCS, S. & BÜHLMANN, P. (2021). Change-point detection for graphical models in the presence of missing values. *J. Comput. Graph. Statist.* **30**, 768–779.
- LU, Z., BANERJEE, M. & MICHAELIDIS, G. (2017). Intelligent sampling for multiple change-points in exceedingly long time series with rate guarantees. *arXiv:1710.07420*. 595
- MAIDSTONE, R., HOCKING, T., RIGAILL, G. & FEARNHEAD, P. (2017). On optimal multiple changepoint algorithms for large data. *Stat. Comput.* **27**, 519–533.
- MATTESON, D. S. & JAMES, N. A. (2014). A nonparametric approach for multiple change point analysis of multivariate data. *J. Amer. Statist. Assoc.* **109**, 334–345.
- MAZUMDER, R. & HASTIE, T. (2012a). Exact covariance thresholding into connected components for large-scale graphical Lasso. *J. Mach. Learn. Res.* **13**, 781–794. 600
- MAZUMDER, R. & HASTIE, T. (2012b). The graphical Lasso: new insights and alternatives. *Electron. J. Stat.* **6**, 2125–2149.
- OLSHEN, A. B., VENKATRAMAN, E. S., LUCITO, R. & WIGLER, M. (2004). Circular binary segmentation for the analysis of array-based DNA copy number data. *Biostatistics* **5**, 557–572. 605
- PADILLA, O. H. M., YU, Y., WANG, D. & RINALDO, A. (2021). Optimal nonparametric change point analysis. *Electron. J. Stat.* **15**, 1154–1201.
- PADILLA, O. H. M., YU, Y., WANG, D. & RINALDO, A. (2022). Optimal nonparametric multivariate change point detection and localization. *IEEE Trans. Inf. Theory* **68**, 1922–1944.
- RIGAILL, G. (2015). A pruned dynamic programming algorithm to recover the best segmentations with 1 to k_{max} change-points. *Journal de la société française de statistique* **156**, 180–205. 610
- ROSENBERG, A. & HIRSCHBERG, J. (2007). V-measure: a conditional entropy-based external cluster evaluation measures. *Proc. Conf. Empir. Meth. Nat. Lang. Process.*, 410–420.
- ROY, S., ATCHADÉ, Y. & MICHAELIDIS, G. (2017). Change point estimation in high dimensional Markov random-field models. *J. R. Statist. Soc. B* **79**, 1187–1206. 615
- SOH, Y. S. & CHANDRASEKARAN, V. (2017). High-dimensional change-point estimation: combining filtering with convex optimization. *Appl. Comput. Harmon. Anal.* **43**, 122–147.
- TIBSHIRANI, R. (1996). Regression shrinkage and selection via the Lasso. *J. R. Statist. Soc. B* **58**, 267–288.
- VERZELEN, N., FROMONT, M., LERASLE, M. & REYNAUD-BOURET, P. (2020). Optimal change-point detection and localization. *arXiv:2010.11470*. 620
- VOSTRIKOVA, L. Y. (1981). Detecting 'disorder' in multidimensional random processes. *Soviet Math. Dokl.* **24**, 55–59.
- WANG, D., LIN, K. Z. & WILLETT, R. (2021). Statistically and computationally efficient change point localization in regression settings. *J. Mach. Learn. Res.* **22**, 1–46.
- WANG, D., YU, Y. & RINALDO, A. (2020). Univariate mean change point detection: Penalization, CUSUM and optimality. *Electron. J. Stat.* **14**, 1917–1961. 625
- WANG, D., YU, Y. & RINALDO, A. (2021). Optimal covariance change point localization in high dimensions. *Bernoulli* **27**, 554–575.
- WANG, T. & SAMWORTH, R. J. (2018). High dimensional change point estimation via sparse projection. *J. R. Statist. Soc. B* **80**, 57–83. 630
- WITTEN, D. M., FRIEDMAN, J. H. & SIMON, N. (2011). New insights and faster computations for the graphical Lasso. *J. Comput. Graph. Statist.* **20**, 892–900.
- YAO, Y.-C. (1988). Estimating the number of change-points via Schwarz' criterion. *Stat. Probab. Lett.* **6**, 181–189.
- ZHANG, N. R. & SIEGMUND, D. O. (2007). A modified Bayes information criterion with applications to the analysis of comparative genomic hybridization data. *Biometrics* **63**, 22–32. 635
- ZHENG, C., ECKLEY, I. A. & FEARNHEAD, P. (2022). Consistency of a range of penalised cost approaches for detecting multiple changepoints. *Electron. J. Stat.* **16**, 4497–4546.

[Received on 14 February 2020. Editorial decision on X X 2021]