Journal of Statistical Software

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

Comparing binsegRcpp with other implementations of binary segmentation for changepoint detection

Toby Dylan Hocking
Northern Arizona University

Abstract

Binary segmentation is a classic algorithm for detecting changepoints in sequential data. In theory, using a simple loss function like Gaussian or Poisson, binary segmentation should be extremely fast for N data and K segments, $O(NK)$ in the worst case, and $O(N \log K)$ in the best case. In practice existing implementations such as ruptures (Python module) and changepoint (R package) are much slower, and in fact sometimes do not return a correct result. We present the R package **binsegRcpp**, which provides an efficient C++ implementation of binary segmentation, and always returns correct results. We discuss some C++ coding techniques which were used to avoid repetition, for efficiency, and for readability. We additionally include detailed theoretical and empirical comparisons with other implementations of binary segmentation in R packages **fpop**, **blockcpd** and **wbs**.

Keywords: JSS, style guide, comma-separated, not capitalized, R.

1. Introduction: previous software for change-point detection

TODO `fpop::multiBinSeg` (Maidstone, Hocking, Rigaill, and Fearnhead 2017).

TODO `wbs::sbs` (Baranowski and Fryzlewicz 2019).

TODO `changepoint::cpt.mean(method="BinSeg")` (Killick and Eckley 2014; Killick, Haynes, and Eckley 2022).

TODO `ruptures.Binseg` (Truong, Oudre, and Vayatis 2020).

TODO `fastcpd` package (Li and Zhang) (nice related work table).

2. Models and software

2.1. C++ features

We used several features of C++ to implement **binsegRcpp** efficiently. In binary segmentation, a segment is split in each iteration, which results in up to two new segments that must be considered to split (see Section **??** for details). Computing the cost of splitting a segment of size N is $O(N \log N)$ time for L1/Laplace losses, and $O(N)$ time for the normal/Poisson losses. After computing the cost for two new segments, the cost of all splittable segments must be considered, in order to identify the min cost segment to split next. We therefore propose using two different kinds of C++ Standard Template Library containers to store and retreive the cost of segments which have previously been considered, but not yet split (Table 2). First, we propose using the `multiset` container (red-black tree), which can be used to achieve best case time complexity. The `multiset` keeps segments sorted by their cost values, so offers constant $O(1)$ time retreival of the min cost segment to split next; insertion takes logarithmic $O(\log n)$ time in the number of segments n currently stored in the container. Second, we propose using the `list` container (doubly-linked list), which is used as a baseline, to demonstrate the benefits of the multiset container. The `list` keeps segments stored in the order in which they were inserted, so offers constant $O(1)$ time insertion of new elements, and linear $O(n)$ time retreival of the min cost segment to split, for a container with n segments currently stored.

A C++ virtual class `Container` is used to represent the methods that must be provided by either type of container: `insert`, `get_best`. We used a C macro, `CMAKER`, to declare the two sub-classes:

```
#define CMAKER(CONTAINER, INSERT, GET_BEST)...
CMAKER(multiset, insert, segment_container.begin())
CMAKER(list, push_back, std::min_element(
 segment_container.begin(),segment_container.end()))
```

The code above is expanded by the C pre-processor, to two declarations of `Container` sub-classes.

Another C macro is used to declare the loss functions which require the cumulative median:

```
#define ABS_DIST(NAME, VARIANCE)...
ABS_DIST(l1, false)
ABS_DIST(laplace, true)
```

The loss function which requires a variance estimate is the Laplace negative log likelihood; the simple L1 loss is obtained without a variance estimate. A `class absDistribution` implements the common operations for the two loss functions, including the cumulative median in $O(N \log N)$ time using the algorithm of Drouin, Hocking, and Laviolette (2017).

Another C macro is used to declare the loss functions which require the cumulative sum:

```
#define CUM_DIST(NAME, COMPUTE, ERROR, VARIANCE)...
CUM_DIST(mean_norm, RSS, , false)
CUM_DIST(poisson,
 (mean>0) ? (mean*N - log(mean)*sum) : ( (sum==0) ? 0 : INFINITY ),
 if(round(value)!=value)return ERROR_DATA_MUST_BE_INTEGER_FOR_POISSON_LOSS;
```

Parameter	Description	Definition
`mean`	model parameter	$(\sum_{i\in\text{subtrain}} w_i x_i)/(\sum_{i\in\text{subtrain}} w_i)$

```
 if(value < 0)return ERROR_DATA_MUST_BE_NON_NEGATIVE_FOR_POISSON_LOSS;,
 false)
CUM_DIST(meanvar_norm,
 (var>max_zero_var) ? (RSS/var+N*log(2*M_PI*var))/2 : INFINITY,
 , true)
```

The code below declares three loss functions based on the cumulative sum: `mean_norm` is the residual sum of squares (RSS), `poisson` loss is for count data, and `meanvar_norm` is the normal model with change in mean and variance. The `COMPUTE` parameter specifies the loss, in terms of six variables: `mean`, `var`, `N`, `sum`, `squares`, `max_zero_var` (Table **??**). These variables are computed in a `class CumDistribution` which contains the logic common to these loss functions (cumulative sum, etc). Note that `mean` and `var` parameters code can compute the loss of held-out validation data (not used to compute model parameters), with respect to a subset TODO. The `ERROR` parameter is an optional block of code, that is used to return an error status code, if there are unusual input data.

Additionally, the `CUM_DIST` and `ABS_DIST` macros generate code that populates a static `unordered_map` which can be queried to obtain a list of distributions which are supported by the C++ code:

```
R> binsegRcpp::get_distribution_info()$dist
```

```
[1] "meanvar_norm" "poisson"      "mean_norm"    "laplace"
[5] "l1"
```

To illustrate the performence benefits of using log-time containers were used to efficiently , virtual classes, static variables, function pointers, templates, macros).

Loss computation. We run each algorithm on `N.data` points up to `max.segments = max.changes+1`. `binsegRcpp::binseg` result is a list which contains a data table with `max.segments` rows and column `loss` that is the square loss. `changepoint::cpt.mean` result is a list of class `cpt.range` with method `logLik` which returns the square loss of one of the models. `wbs::sbs` result is a list which contains a data frame with `N.data-1` rows and `CUSUM` and `min.th` columns TODO. `fpop::multiBinSeg` result is a list with element `J.est`, which is a vector of `max.changes` square loss decrease values. `ruptures.Binseg.predict` result is a vector of segment ends for one model size, which can be passed to `sum_of_costs` method to compute the square loss.

Three packages have implemented the normal change in mean and variance model. `binsegRcpp::binseg` loss values are the normal negative log likelihood (NLL). The `changepoint::logLik` function returns two times the NLL. The `ruptures.Binseg` loss is related via this equation,

$$\text{NLL} = (\text{rupturesLoss} + N[1 + \log(2\pi)])/2 \tag{1}$$

package	binsegRcpp	changepoint	wbs	fpop	ruptures
function	binseg	cpt.mean	sbs	multiBinSeg	Binseg
version	2022.3.29	2.2.3	1.4	2019.8.26	1.1.6
weights	yes	no	no	no	no
max segs	yes	yes	no	yes	yes
dim	one	one	one	multi	multi
correct	yes	no	yes	yes	no
losses	C++	C	L2	L2	Python
storage	STL multiset	arrays	recursion	heap	LRU cache
space	$O(S)$	$O(S^2)$	$O(S)$	$O(S)$	$O(S)$
cumsum	yes	yes	yes	yes	no
best	$O(N \log N)$	$O(N^3)$	$O(N \log N)$	$O(N \log N)$	$> O(N \log N)$
worst	$O(N^2)$	$O(N^3)$	$O(N^2)$	$O(N^2)$	$O(N^2)$
CV	yes	no	no	no	no
params	yes	largest	no	no	no

Table 1: TODO

case/splits:	one		best/equal		worst/unequal	
operation:	insert	argmin	insert	argmin	insert	argmin
list	$O(1)$	$O(n)$	$O(S \log S)$	$O(S)$	$O(S)$	$O(S)$
heap/multiset	$O(\log n)$	$O(1)$	$O(S)$	$O(S^2)$	$O(S)$	$O(S)$
Find new split			$O(N \log S)$		$O(NS)$	

Table 2: container properties

3. Illustrations

TODO

```
R> data("quine", package = "MASS")
```

TODO

4. Summary and discussion

- As usual ...

References

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

Drouin A, Hocking T, Laviolette F (2017). "Maximum Margin Interval Trees." In I Guyon, UV Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, R Garnett (eds.), *Advances*

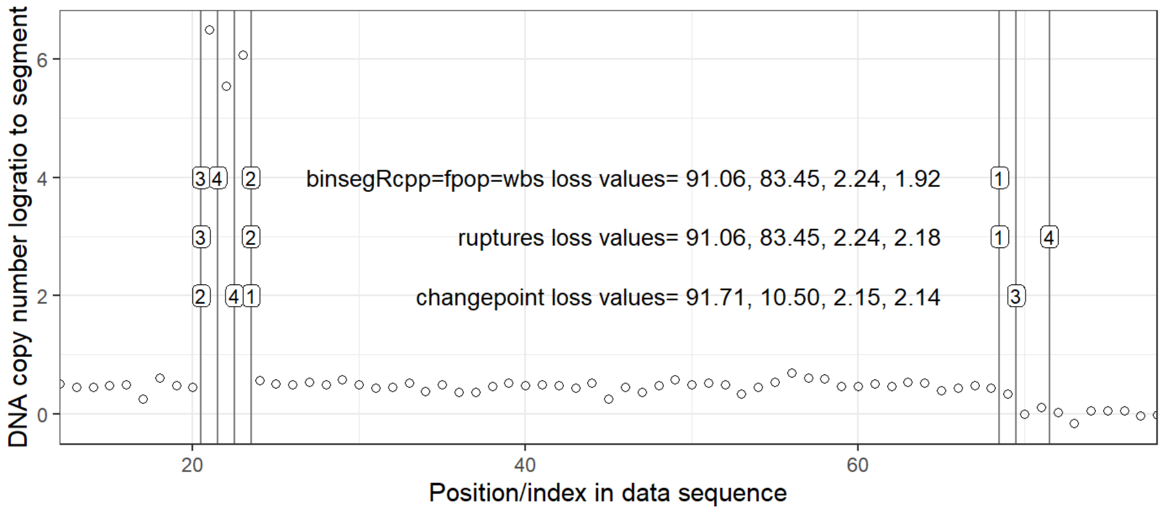

Figure 1: For a real cancer DNA copy number data set with 273 observations, we show the first four changepoints detected by several different implementations of binary segmentation.

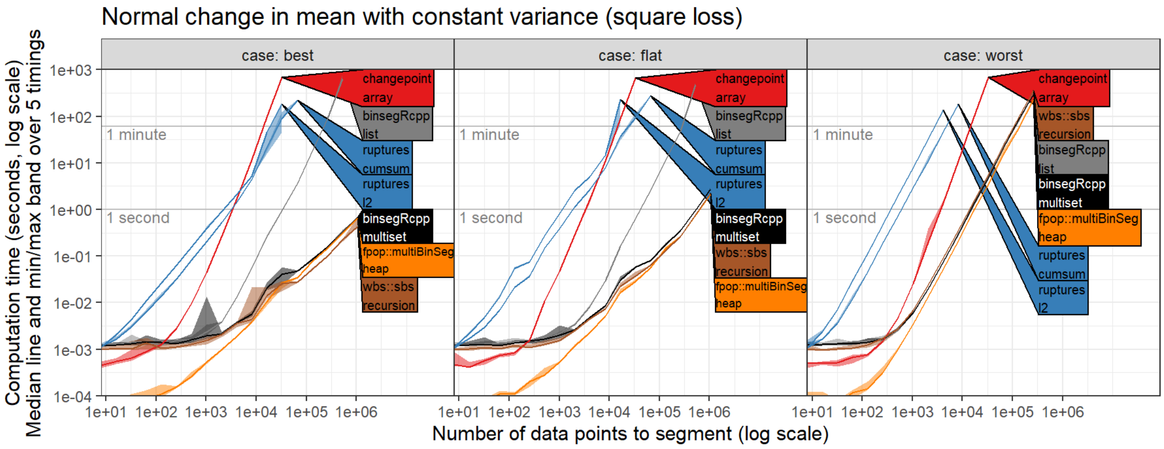

Figure 2: Timings using square loss.

in Neural Information Processing Systems 30, pp. 4947–4956. Curran Associates, Inc. URL http://papers.nips.cc/paper/7080-maximum-margin-interval-trees.pdf.

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

Killick R, Haynes K, Eckley IA (2022). *changepoint: An R package for changepoint analysis.* R package version 2.2.3, URL https://CRAN.R-project.org/package=changepoint.

Li X, Zhang X (????). "fastcpd: Fast Change Point Detection in R." Pre-print arXiv:2404.05933.

Maidstone R, Hocking T, Rigaill G, Fearnhead P (2017). "On optimal multiple changepoint

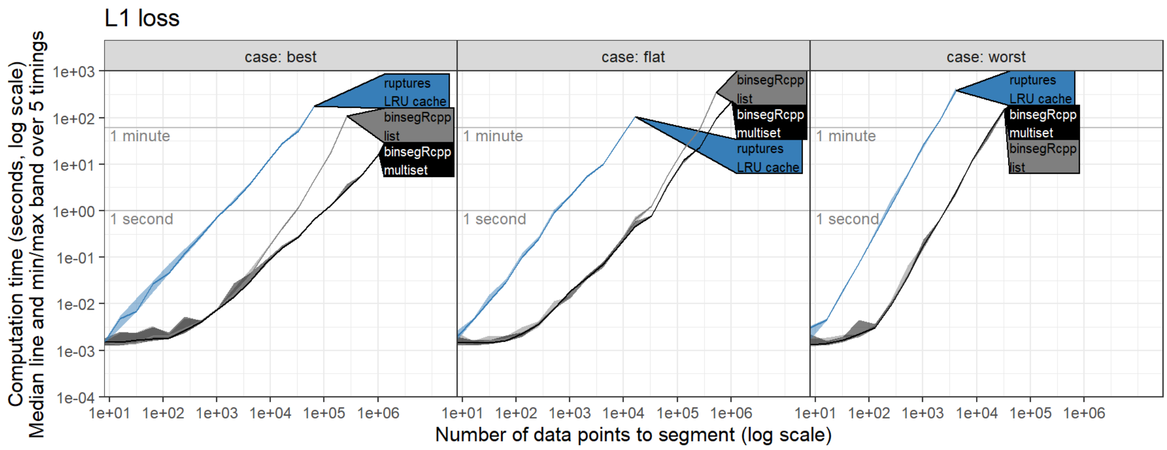

Figure 3: Timings using L1 loss.

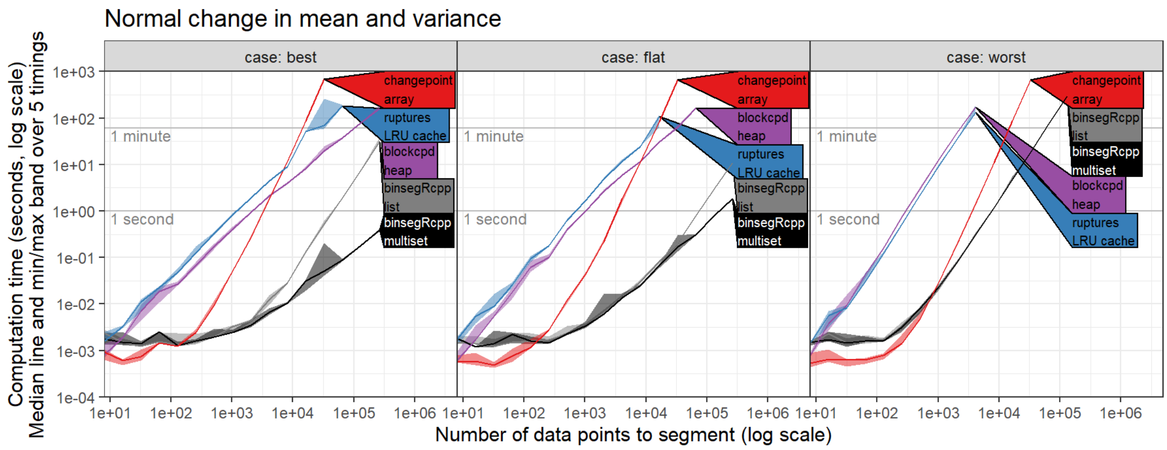

Figure 4: Timings using Gaussian change in mean and variance model.

algorithms for large data." *Statistics and Computing*, **27**. URL https://link.springer.com/article/10.1007/s11222-016-9636-3.

Truong C, Oudre L, Vayatis N (2020). "Selective review of offline change point detection methods." *Signal Processing*, **167**, 107299. ISSN 0165-1684. doi:https://doi.org/10.1016/j.sigpro.2019.107299. URL https://www.sciencedirect.com/science/article/pii/S0165168419303494.

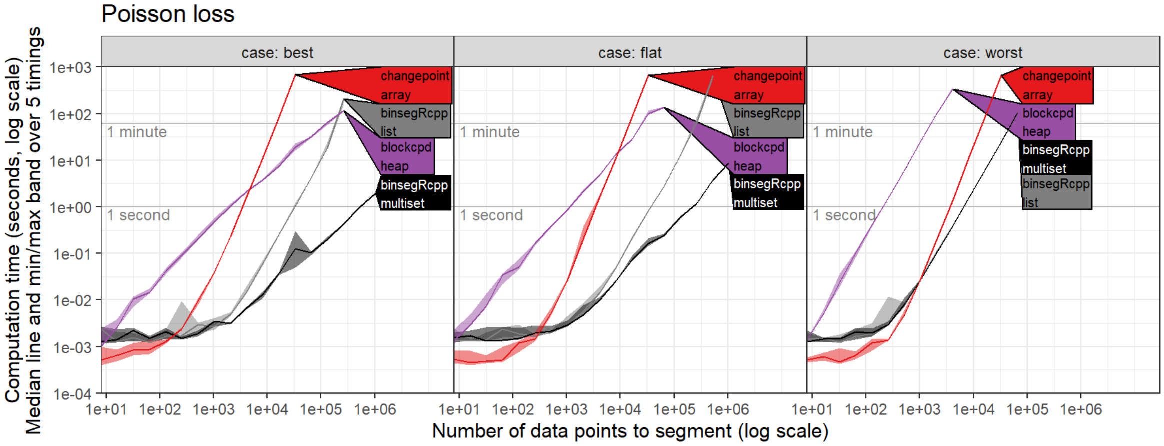

Figure 5: Timings using Poisson loss.

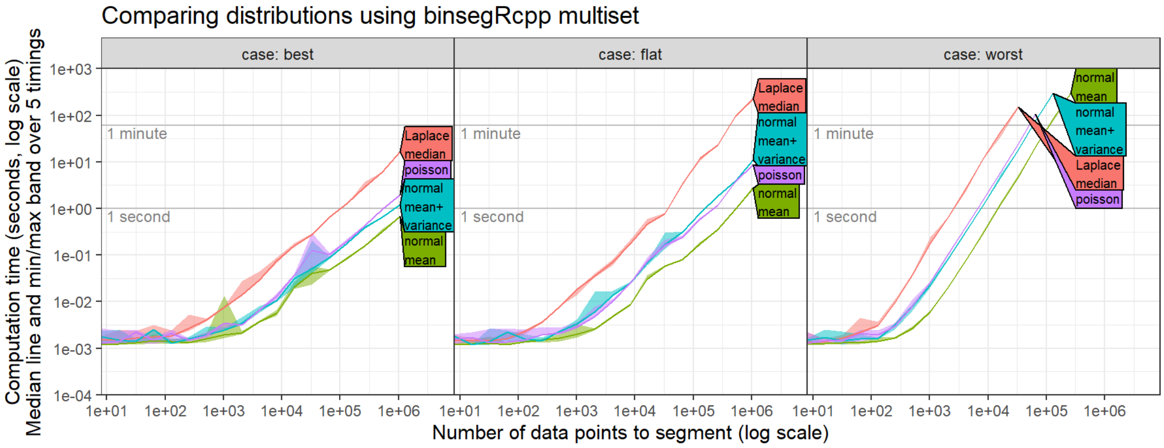

Figure 6: Timings using binsegRcpp multiset with several different loss functions.

A. More technical details

TODO

Affiliation:

Toby Dylan Hocking
Université de Sherbrooke
E-mail: Toby.Hocking@R-project.org

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