# Fast and Parallel Algorithm for Population-Based Segmentation of Copy-Number Profiles

Guillem Rigaill[1(✉)], Vincent Miele[2], and Franck Picard[2]

[1] Unité de Recherche en Génomique Végétale (URGV) INRA-CNRS-Université d'Evry Val d'Essonne, 2 Rue Gaston Crémieux, 91057 Evry Cedex, France
rigaill@evry.inra.fr
[2] Laboratoire de Biométrie et Biologie Evolutive, UMR CNRS 5558 Université Lyon 1, 69622 Villeurbanne, France
{vincent.miele,franck.picard}@univ-lyon1.fr

**Abstract.** Dynamic Programming (DP) based change-point methods have shown very good statistical performance on DNA copy number analysis. However, the quadratic algorithmic complexity of DP has limited their use on high-density arrays or next generation sequencing data. This complexity issue is particularly critical for segmentation and calling of segments, and for the joint segmentation of many different profiles. Our contribution is two-fold. First we provide an at worst linear DP algorithm for segmentation and calling, which allows the use of DP-based segmentation on high-density arrays with a considerably reduced computational cost. For the joint segmentation issue we provide a parallel version of the `cghseg` package which now allows us to analyze more than 1,000 profiles of length 100,000 within a few hours. Therefore our method and software package are adapted to the next generation of computers (multi-cores) and experiments (very large profiles).

**Keywords:** DNA copy number · Dynamic Programming · Segmentation · Joint segmentation · Parallel computing

## 1 Introduction

Segmenting heterogeneous signals into regions of common characteristics is often required for biologists that face high dimensional information. This partitioning helps reducing the dimension of the data and provides guidelines for interpretation and further biological investigation. Such methods have been widely applied in Genomics, to unravel DNA sequences structures using base composition [6], to segment expression profiles [2,7], and to determine copy-number variations based on array CGH data. Microarray CGH data analysis is certainly the field for which every possible method of segmentation have been tried. Three main categories of methods lead to many developments: Circular Binary Segmentation (CBS, [8]), Hidden Markov Models [5], and change-point analysis [10,11]. Our purpose


E. Formenti et al. (Eds.): CIBB 2013, LNBI 8452, pp. 248–258, 2014.
DOI: 10.1007/978-3-319-09042-9_18

in this work is not to compare the different methods for array CGH analysis. Such comparisons have been done elsewhere [18], and extensive reviews now exist on the subject [14,16]. Our focus is the following: in every comparative study, change-point analysis based on Dynamic Programming (DP) has shown excellent performance along with CBS [3,18]. The advantage of DP-based segmentation is that it provides the best global segmentation, i.e. the segmentation that globally minimizes a likelihood criterion, whereas local approaches like CBS only provide local minimizers. Moreover, change-point models can also integrate a calling step that is used to cluster segments that show the same copy-number on average (also called segmentation/clustering) [11,17], which has been shown to be of central importance in the segmentation process [18]. Unfortunately, the algorithmic complexity of DP is proportional to the square of the signal's length, which has hampered the use of such method on high-density arrays for instance. This issue has become even more problematic when dealing with population-based or joint segmentation [9,15,20]. Our contribution is two-fold: first we provide a linearized version of the DP algorithm for segmentation/clustering adapting pruning strategies that have recently emerged in the field [4,13]. Secondly we deal with the joint segmentation issue by providing a parallel version of existing algorithms implemented in the `cghseg` package. With the growing availability of multicore computers (from laptops to many-core servers), it has become essential to provide software that use every available resource. `R` has been a tremendous platform for package distribution, and here we provide a new version of the `cghseg` package, a *next generation package* that is adaptive to available computing power. The performance of `cghseg` are impressive: segmenting 1,000 profiles of length 100,000 can now be done in few hours, which was impossible before. This makes segmentation models a new exact investigation method that can be used in routine for exploratory as well as deep analysis.

## 2 Linearization of Dynamic Programming for Segmentation and Segmentation/Clustering

The purpose of segmentation is to partition a signal of $n$ observations $\{Y(t)\}$ into $K$ segments of homogeneous distributional parameter. In this section we deal with the univariate segmentation of one array CGH profile, the case of Joint multivariate segmentation being considered in Sect. 3. In the following a segment is an interval delimited by two change-points $\tau_k, \tau_{k+1}$ for instance. $r_k = [\![\tau_k, \tau_{k+1} - 1]\!]$ stands for segment $k$. To stick to this definition for the first and last segment we use the convention $\tau_0 = 1$ and $\tau_K = n + 1$. A segmentation in $K$ segments is denoted by $m^{(K)} = \{r_0, r_1, \ldots, r_{K-1}\}$.

A standard statistical model for segmentation is the detection of changes in the mean of a signal, such that $\forall t \in r_k$, $Y(t) \sim \mathcal{N}(\mu_k, \sigma^2)$. In the special case of array CGH, an additional step is the "calling" step that is performed by introducing additional (hidden) label variables $\{Z(r_k)\}_k$ to cluster each segment

into categories such as `deleted`, `normal`, `amplified` (not limited to 3 states). Then the segmentation/clustering model becomes:

$$\forall t \in r_k,\ \forall p \in \{1...P\},\ Y(t)|\{Z(r_k) = p\} \sim \mathcal{N}(\mu_p, \sigma^2),$$

with $P$ the total (fixed) number of hidden states, with $\pi_p$ the proportion of segments in each state.

Once the statistical model has been defined, the main algorithmic challenge lies in the exact determination of the boundaries of segments $\{\tau_k\}$ (and not in the estimation of mean parameters $\mu_k$s or $\mu_p$s depending on the model). A well known solution to this problem is to use Dynamic Programming for a given number of segments $K$ to find the best *global* segmentation in terms of "cost" (to be defined). In this work, we do not deal with the issue of model selection to estimate $K$ and $P$, discussed elsewhere [10,19]. To perform Dynamic Programming, we need to define the "unit" cost of a generic segment $r = [\![t_1, t_2]\!]$, which is given by minus the local log-likelihood calculated on $r$:

$$C^{(1)}_{(r)} = \begin{cases} \sum_{t\in r}(y(t) - \mu_r)^2/2\sigma^2, \text{ for segmentation in the mean} \\ -\log\left(\sum_p \pi_p \exp\left\{-\sum_{t\in r}(y(t) - \mu_p)^2/2\sigma^2\right\}\right), \text{ for seg/clust,} \end{cases}$$

with superscript (1) standing for "one segment". A main difference between the two models lies in the estimation of the mean parameters. In the case of segmentation in the mean, parameters $\{\mu_k\}_k$ can be estimated directly by the empirical means of segments while computing the position of the breaks. In the case of segmentation/clustering, parameters $\{\mu_p\}_p$ are common across segments. Consequently, they are fixed while computing breakpoint coordinates, and they are estimated iteratively by using an EM-algorithm, leading to a so-called DP-EM algorithm [11]. In the case of segmentation/clustering, we propose to simplify the cost function by using a *classification cost function*, an approximation denoted by $\widetilde{C}^{(1)}_{(r)}$ which consists in focusing on the dominant term within the sum over $P$ exponentials:

$$\widetilde{C}^{(1)}_{(r)} = \min_p \left\{ \sum_{t\in r} \frac{(y(t) - \mu_p)^2}{2\sigma^2} + \log(\pi_p) \right\}. \tag{1}$$

This cost function is analog to the cost function that is used in standard k-means algorithms.

Since the purpose is to find the global minimum of the total cost function into $K$ segments, we also introduce the set of all segmentations of a given segment $r$ into $K$ segments such that $\mathcal{M}^{(K)}_{(r)} = \mathcal{M}^{(K)}_{t_1,t_2}$. Then the optimal cost of a segmentation of $r$ into $K$ segments and its associated optimal segmentation are defined as:

$$C^{(K)}_{(r)} = \min_{m\in\mathcal{M}^{(K)}_{(r)}} \left\{ \sum_{r\in m} C^{(1)}_{(r)} \right\}, \text{ and } \widehat{m}^{(K)}_{(r)} = \operatorname*{argmin}_{m\in\mathcal{M}^{(K)}_{(r)}} \left\{ \sum_{r\in m} C^{(1)}_{(r)} \right\}.$$

Similarly, when Approximation $\widetilde{C}^{(1)}_{(r)}$ is used (Eq. (1)), we use notations $\widetilde{C}^{(K)}_{(r)}$ and $\widetilde{m}^{(K)}_{(r)}$. When the cost of a segmentation is segment additive (which is the case

in both models), a $\mathcal{O}(Kn^2)$ Dynamic Programming algorithm can be built to recover the best exact segmentation (into 1 to $K$ segments).

### 2.1 Original Dynamic Programming Algorithm for Segmentation

A basic statement is that the cost of a given segmentation is the sum of the cost of its segments. Thus the Bellman optimality principal holds and we have: $C_{1,t}^{(k+1)} = \min_{\tau \leq t}\{C_{1,\tau-1}^{(k)} + C_{\tau,t}^{(1)}\}$. Using this update rule a Dynamic Programming algorithm can be built to recover all $C_{1,t}^{(k+1)}$ for all $t \leq n$ and $k \leq K$. This can be done using Algorithm 1 for instance. For simplicity we did not include the initialization of all $C_{1,t}^{(1)}$ for $t \leq n$ and of all $C_{1,t}^{(k)}$ for $k \leq K$ and $t < k$. All $C_{1,t}^{(k)}$ are initialized as $+\infty$ and $C_{1,t}^{(1)}$ are initialized using their definition. This algorithm assumes that all $C_{t_1,t_2}^{(1)}$ have been pre-computed and stored (in a $n$ by $n$ matrix) or that they can be efficiently computed on the fly, which is the case for every models we consider here. At step $k, t$ of Algorithm 1 $\mathcal{O}(t)$ basic operations are performed. If we sum these for all $k < K$ and $t < n$ we see that the algorithm has a $\mathcal{O}(Kn^2)$ time complexity. This $n^2$ factor is the main reason why Dynamic Programming can be prohibitive to use on large signals (like SNP arrays for instance).

**Algorithm 1.** Standard DP algorithm

**Input**: $Y(t)$ a profile of $n$ observations, $K$ an integer
**Output**: $C_{1,t}^{(k)}$ in $\mathbb{R}$ and $M_{1,t}^{(k)}$ in $\mathbb{N}$ for all $k \leq K$ and $t \leq n$
**for** $t \in [\![1, n]\!]$ **do**
  $C_{1,t}^{(1)} = C_{1t}$ ; $M_{1,t}^{(1)} = 0$
**end for**
**for** $k \in [\![2, \min(t, K)]\!]$ **do**
  **for** $t \in [\![1, n]\!]$ **do**
    $C_{1,t}^{(k)} = \min_{k-1 \leq \tau \leq t-1}\{C_{1,\tau}^{(k-1)} + C_{(\tau+1)t}^{(1)}\}$ ; $M_{1,t}^{(k)} = \operatorname{argmin}_{k-1 \leq \tau \leq t-1}\{C_{1,\tau}^{(k-1)} + C_{(\tau+1)t}^{(1)}\}$
  **end for**
**end for**

### 2.2 A Linear Dynamic Programming Algorithm for the Classification Cost Function

An important consequence of using the classification cost $\widetilde{C}_{(r)}^{(1)}$ rather than $C_{(r)}^{(1)}$ is that the set of candidate segmentations can be pruned efficiently. The idea of pruning the set of candidate segmentations is not new and was proposed for other cost functions [4,13]. In these two algorithms the pruning step usually allows for an important speed up and the average time complexity is for many signal in $\mathcal{O}(n)$ or $\mathcal{O}(n \log(n))$. Nonetheless, in both cases, the worst case is quadratic with

respect to $n$. In the case of the classification cost however the pruning step is particularly efficient and we can guarantee that the time and space complexity of the algorithm are at worst in $\mathcal{O}(KPn)$. Furthermore, we can also guarantee that the average cost (over all data points) of the recovered segmentation $\widetilde{m}_{(r)}^{(K)}$ is at worst within $K\log(P)/n$ of the optimal segmentation $\widehat{m}_{(r)}^{(K)}$ (see Theorem 22).

Before we describe the algorithm we need to define some new notations. We define the approximate cost of a segment knowing its mean $\mu$ as $\widetilde{C}_{(r)}^{(1)}(\mu) = \left\{\sum_{t\in r}(y(t)-\mu)^2/2\sigma^2 - \log(\pi_p)\right\}$. Using this notation we can rewrite the classification cost of a segment $r = [\![t_1, t_2]\!]$ as $\widetilde{C}_{t_1,t_2}^{(1)} = \min_p\{\widetilde{C}_{t_1,t_2}^{(1)}(\mu_p)\}$, with $\widetilde{C}_{t_1,t_2}^{(1)}(\mu)$ being point additive in the sense that $\widetilde{C}_{t_1,t_2}^{(1)}(\mu) = \sum_{t_1\le t\le t_2} \widetilde{C}_{t,t}^{(1)}(\mu)$. Then we define the cost of the best segmentation knowing that the mean of the last segment is $\mu$ as:

$$\widetilde{C}_{1,t}^{(K)}(\mu) = \min_{m\in\mathcal{M}_{(1,t)}^{(K)}} \left\{ \sum_{k<K-1} \widetilde{C}_{(r_k)}^{(1)} + \widetilde{C}_{(r_K)}^{(1)}(\mu) \right\}.$$

Using this notation we get that $\widetilde{C}_{1,t}^{(k)} = \min_{p<P}\left\{\widetilde{C}_{1,t}^{(k)}(\mu_p)\right\}$, and if we know every $C_{1,t}^{(k)}(\mu_p)$ at step $t$ for all $p < P$, we straightfowardly get $C_{1,t}^{(k)}$ in $\mathcal{O}(p)$. As $\widetilde{C}_{t_1,t_2}(\mu)$ is point additive updating $\widetilde{C}_{t_1,t_2}(\mu)$ is easy and can be done efficiently using the following theorem.

**Theorem 21.** $\widetilde{C}_{1,t+1}^{(k)}(\mu) = \min\left\{\widetilde{C}_{1,t}^{(k)}(\mu), \widetilde{C}_{1,t}^{(k-1)}\right\} + \widetilde{C}_{t+1,t+1}^{(1)}(\mu)$

*Proof.* Let us first notice that: $\widetilde{C}_{1,t+1}^{(k)}(\mu) = \min_{\tau<t+1}\left\{\widetilde{C}_{(1,\tau)}^{(k-1)} + \widetilde{C}_{(\tau+1,t)}^{(1)}(\mu)\right\}$. Using the definition of $\widetilde{C}_{1,t}^{(k)}(\mu)$ we get that:

$$\widetilde{C}_{1,t}^{(k)}(\mu) + \widetilde{C}_{(t+1,t+1)}^{(1)}(\mu) = \min_{\tau<t}\left(\widetilde{C}_{(1,\tau)}^{(k-1)}\right) + \widetilde{C}_{(t+1,t+1)}^{(1)}(\mu)$$

From this the theorem follows. ∎

Using this theorem, knowing $\widetilde{C}_{1,t}^{(k)}(\mu)$ and $\widetilde{C}_{1,t}^{(k-1)}$ we get $\widetilde{C}_{1,t+1}^{(k)}(\mu)$ in $\mathcal{O}(1)$ and we derive Algorithm 2 for the DP step of the DP-EM algorithm [11]. For simplicity we did not include the initialization of $\widetilde{C}_{1,t}^{(1)}$ for $t < n$ and $\widetilde{C}_{1,1}^{(k)}(\mu_p)$. All $\widetilde{C}_{1,1}^{(k)}(\mu_p)$ are initialized as $+\infty$ and $\widetilde{C}_{1,t}^{(1)}$ are initialized using their definition. At step $k, t$ of Algorithm 2 $\mathcal{O}(P)$ basic operations are performed. If we sum these for all $k < K$ and $t < n$ we straightforwardly see that the algorithm has an $\mathcal{O}(KPn)$ time complexity.

### 2.3 A Bound on the Quality of the Approximation

Using the approximation defined in Eq. 1 we can guarantee the quality of the obtained segmentation using the following theorem.

**Algorithm 2.** Linear DP algorithm for the classification cost

**Input**: $Y(t)$ a profile of $n$ observations, $K$ an integer
**Input**: $C_{t_1,t_2}$ cost of the segments $[\![t_1, t_2]\!]$ for all $(t_1, t_2) \in [\![1, n]\!]^2$
**Output**: $C_{1,t}^{(k)}$ in $\mathbb{R}$ and $M_{1,t}^{(k)}$ in $\mathbb{N}$ for all $k \leq K$ and $t \leq n$
**for** $k \in [\![2, \min(t, K)]\!]$ **do**
  **for** $t \in [\![1, n]\!]$ **do**
    **for** $p \in [\![1, P]\!]$ **do**
      $C_{1,t}^{(k)}(\mu_p) = \min\{C_{1,t-1}^{(k)}(\mu_p), C_{1,t-1}^{(k-1)}\} + C_{t,t}^{(1)}(\mu_p)$ ;
      $M_{1,t}^{(k)}(\mu_p) = \operatorname{argmin}\{C_{1,t-1}^{(k)}(\mu_p), C_{1,t-1}^{(k-1)}\}$
    **end for**
    $C_{1,t}^{(k)} = \min_p\{C_{1,t}^{(k)}(\mu_p)\}$ ; $p^* = \operatorname{argmin}_p\{C_{1,t}^{(k)}(\mu_p)\}$ ; $M_{1,t}^{(k)} = M_{1,t}^{(k)}(\mu_{p^*})$
  **end for**
**end for**

**Theorem 22.** *Using approximation defined in Eq. 1 we have for all segments $R$ and $K$*

$$\widetilde{C}_{(R)}^{(K)} - K\log(P) \;\leq\; \sum_{r \in \widehat{m}_{(R)}^{(K)}} \widetilde{C}_{(R)}^{(K)} - K\log(P) \;\leq\; C_{(R)}^{(K)} \;\leq\; \sum_{r \in \widetilde{m}_{(R)}^{(K)}} C_{(R)}^{(K)} \;\leq\; \widetilde{C}_{(R)}^{(K)}.$$

*Proof.* We have $C_{(r)}^{(1)} = -log(\sum_p \exp\{-\widetilde{C}_{(r)}^{(1)}(\mu_p)\})$ and $\widetilde{C}_{(r)}^{(1)} \geq \widetilde{C}_{(r)}^{(1)}(\mu_p)$. From this we get that: $\forall r$, $\widetilde{C}_{(r)}^{(1)} - \log(P) \leq C_{(r)}^{(1)} \leq \widetilde{C}_{(r)}^{(1)}$, which gives, along with the definition of $C_{(R)}^{(K)}$:

$$\forall m \in \mathcal{M}_{(R)}^{(K)},\ \widetilde{C}_{(R)}^{(K)} - K\log(P) \leq \sum_{r \in m} \widetilde{C}_{(r)}^{(1)} - K\log(P) \leq \sum_{r \in m} C_{(r)}^{(1)}. \tag{2}$$

Similarly we get:

$$\forall m \in \mathcal{M}_{(R)}^{(K)},\ C_{(R)}^{(K)} \leq \sum_{r \in m} C_{(r)}^{(1)} \leq \sum_{r \in m} \widetilde{C}_{(r)}^{(1)}. \tag{3}$$

Applying Eq. 2 to $m = \widehat{m}_{t_1,t_2}^{(K)}$ and then Eq. 3 to $m = \widetilde{m}_{t_1,t_2}^{(K)}$ we get the theorem. ∎

## 3 Joint Segmentation and Parallelization of the Algorithm

Joint segmentation arises when more than one profile should be segmented jointly. We make the distinction between simultaneous segmentation, where all breakpoints are the sames across profiles, with joint segmentation where all profiles have their own specific breakpoints, but may share some characteristics, like the same noise, the same biases, the same values for the mean of segments that

share the same copy numbers (i.e. parameters $\mu_p$). When segmenting $I$ profiles jointly, a typical model is:

$$\forall i \in [1, I],\ \forall t \in r_k^i,\ Y_i(t) | \{Z(r_k^i) = p\} \sim \mathcal{N}(\mu_p + b(t), \sigma^2),$$

where $r_k^i$ stands for segment $k$ of profile $i$, and where $b(t)$ is a bias function that depends on the position (like the wave effect [9]). Then the segmentation of profile $i$ into $K_i$ segments is denoted by $m_i^{K_i} = \{r_1^i, ..., r_{K_i}^i\}$, and the *global* segmentation into $K$ segments is denoted by $m_K = \{m_1^{K_1}, ..., m_I^{K_I}\}$, with $K = \sum_{i=1}^{I} K_i$.

Joint segmentation presents an additional algorithmic challenge. When each $K_i$ is known, the best segmentation for each profile $\widehat{m}_i^{K_i}$ can be found independently and therefore computed in parallel using Algorithm 1 or 2. Then the additional step, which remains sequential, is to determine (i) the common parameters across profiles ($\{\mu_p\}, b(t)$) and (ii) the best combination of $\{\widehat{K}_i\}$ that provides the best joint segmentation for a given total number of segments $K$ [9]. Dynamic Programming has also been shown to provide an exact solution to problem (ii) in $\mathcal{O}(I^3 K_{\max}^2)$, where $K_{\max}$ is the maximum number of segments to be put in each profile. Consequently, DP-based joint segmentation alternates between parallel steps (computation of individual segmentations) and sequential steps (estimation of common parameters and determination of the best combination of individual segmentations), as shown in Algorithm 3.

**Algorithm 3.** Parallel Algorithm for Joint segmentation

**Input**: $\{Y_i(t)\}$, $I$ profiles of $n$ observations, $K$ an integer, $\{\widehat{\mu}_p^0\}$ starting values for common parameters
**Input**: $C_{t_1,t_2}^i$ cost of the segments $[\![t_1, t_2]\!]$ for profile $i$ for all $(t_1, t_2) \in [\![1, n]\!]^2$
**Output**: $\{\widehat{K}_1, ..., \widehat{K}_I\}$, $C_{1,t}^{(\widehat{K}_i)}$ in $\mathbb{R}$ for all $i \leq I$, $k \leq K_i$ and $t \leq n$, $\{\widehat{\mu}_p\}$, $\widehat{b}(t)$
**while** not convergence **do**
  **for** $i \in \{1, ...I\}$ **do in parallel**
    compute $C_{1,t}^{(k)}$ $\forall k \leq K$ and $t \leq n$ with Algorithm 2
  **end for**
  update $\{\widehat{\mu}_p\}, \widehat{b}(t)$
  compute $\{\widehat{K}_1, ..., \widehat{K}_I\}$ with **sequential** DP [9]
**end while**

## 4 Correctness, Computational Footprint and Scalability

All the presented algorithms are available into the `cghseg` `R`-package[1], which now relies on the `parallel` `R`-package and on the shared memory programming

[1] http://cran.r-project.org/web/packages/cghseg

**Table 1.** Performance comparison between the non linearized and non parallel ("old") and the linearized-parallel ("new") versions. Performance is assessed through Mean Square Errors (over 50 replicates) for the estimates of the number of segments ($\hat{K}$), the mean level of each segment ($\hat{\mu}$), and confidence intervals for the False Discovery and False Negative Rates for breakpoint detection. The methodology is similar to [9].

| SNR | $\text{MSE}_{\text{new}}(\widehat{\text{K}})$ | $\text{MSE}_{\text{old}}(\widehat{\text{K}})$ | $\text{MSE}_{\text{new}}(\hat{\mu})$ | $\text{MSE}_{\text{old}}(\hat{\mu})$ | $\text{CI}_{\text{new}}(\text{fdr})$ | $\text{CI}_{\text{old}}(\text{fdr})$ | $\text{CI}_{\text{new}}(\text{fnr})$ | $\text{CI}_{\text{old}}(\text{fnr})$ |
|---|---|---|---|---|---|---|---|---|
| 1 | [0.90; 1.02] | [0.90; 1.02] | [0.14; 0.16] | [0.14; 0.17] | [0.42; 0.45] | [0.42; 0.45] | [0.59; 0.62] | [0.59; 0.62] |
| 5 | [0.36; 0.44] | [0.37; 0.45] | [0.05; 0.06] | [0.05; 0.06] | [0.15; 0.20] | [0.17; 0.22] | [0.21; 0.26] | [0.23; 0.28] |
| 10 | [0.22; 0.30] | [0.23; 0.30] | [0.03; 0.03] | [0.03; 0.03] | [0.06; 0.10] | [0.07; 0.11] | [0.08; 0.14] | [0.09; 0.14] |
| 15 | [0.17; 0.22] | [0.17; 0.23] | [0.02; 0.02] | [0.02; 0.02] | [0.02; 0.06] | [0.03; 0.07] | [0.03; 0.08] | [0.03; 0.09] |
| 20 | [0.15; 0.20] | [0.15; 0.21] | [0.01; 0.02] | [0.01; 0.02] | [0.01; 0.05] | [0.01; 0.05] | [0.01; 0.06] | [0.01; 0.07] |

standard `openMP` in the `C++` sections. The new package is now designed to be executed in parallel on multiprocessor architectures.

We recall that the statistical performance of the model have been discussed elsewhere [9], so that our focus here is computational only. Consequently, we use a previously published simulation scheme to generate the data [9,12] with $n = 20{,}000$ observations per profile, and a number of profiles of 256, 512 and 1024. The average number of segments is set to 10 for each profile, and the Signal to Noise Ratio is set to 5 (which corresponds to moderately easy configurations [9]).

We first check the correctness of our method, by verifying that the linearization approximations (with and without calling) give the same statistical performance compared with the non-linearized version. This is shown in Table 1), as the performance are identical over 50 replicates.

Then we assess the effectiveness of the parallel implementation by running `cghseg` on an increasing number of cores $(1, 2, 4, 16, 32, 48)$. When dealing with parallelized codes, the Amdahl's law [1] provides the expected theoretical speedup with respect to the time proportion of the sequential part and the number of cores. The speedup of `cghseg` follows the Amdahl's law when the number of profiles is high (Fig. 1, dashed lines), which demonstrates the quality of our implementation as the Amdahl's law constitutes the best possible speedup. However, we observe an unexpected moderate speedup decrease for a lower number of profiles. This is due to overheads associated with the use of the `parallel` `R`-package, overheads which become negligible when the number of profiles is high. Still, the execution time of configurations with 256 profiles is 6 min on average using 48 cores, which remains excellent.

While the main interest of `cghseg` lies in the quality of its results, the associated computational expense is affordable even for very large datasets. As a benchmark, we simulated datasets with 1024 profiles of length 100,000 which corresponds to the up-to-date limits of the available datasets of SNP-arrays for instance. When joint segmentation is performed along with calling, `cghseg` required 4 h on average on 48 cores (see Table 2), which is very reasonable considering the size of the dataset. Lastly, due to the *copy-on-write* mechanism of the `parallel` `R`-package that avoids memory copy between processes, and thanks to the shared memory efficiency provided by `openMP`, the memory needs of `cghseg`

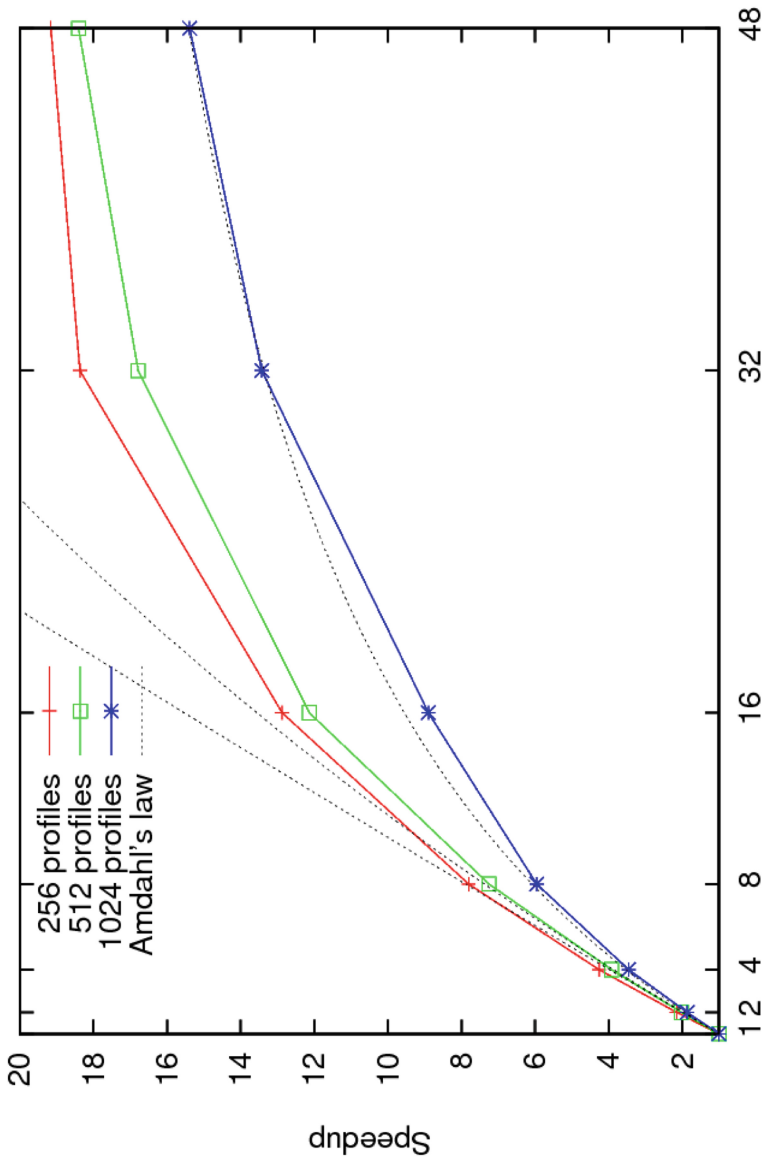


**Fig. 1.** Average speedup of `cghseg` observed on simulated datasets for a varying number of profiles and available cores (plain lines). Amdahl's law is empirically adjusted to the computing results with 256, 512 and 1024 profiles (dashed lines, with $\alpha = 0.2, 1.2, 4.5\,\%$ respectively). The theoretical speedup is given by $1/(\alpha + (1-\alpha)/c)$, where $\alpha$ is the time proportion of the sequential part and $c$ the number of cores. Points correspond to averages over 5 replicates. Run on a quadri-12 cores Opteron 2.2 GHz, 256 Gb RAM.

**Table 2.** Average computational requirements of `cghseg` estimated on simulated datasets for a varying number of profiles and observations, computed on 48 cores. Results correspond to averages over 5 replicates. Run on a quadri-12 cores Opteron 2.2 GHz, 256 Gb RAM.

| $n$ (observations/profile) | 20,000 | | | 100,000 | | |
|---|---|---|---|---|---|---|
| $I$ (number of profiles) | 256 | 512 | 1024 | 256 | 512 | 1024 |
| Average CPU time (min) | 6 | 15 | 54 | 31 | 70 | 253 |
| Memory usage (Gb) | 0.4 | 0.8 | 1.8 | 1.7 | 3.7 | 7.9 |

only correspond to the dataset under study (see Table 2). Therefore, our method and software package are adapted to the next generation of computers (many cores) and experiments (large profiles).

## References


1. Amdahl, G. M.: Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the AFIPS '67 Spring Joint Computer Conference, 18–20 April 1967 (Spring), pp. 483–485. ACM (1967)
2. David, L., Huber, W., Granovskaia, M., Toedling, J., Palm, C.J., Bofkin, L., Jones, T., Davis, R.W., Steinmetz, L.M.: A high-resolution map of transcription in the yeast genome. Proc. Natl. Acad. Sci. USA **103**(14), 5320–5325 (2006)
3. Hocking, T.D., Schleiermacher, G., Janoueix-Lerosey, I., Delattre, O., Bach, F., Vert, J.-P.: Learning smoothing models using breakpoint annotations. HAL Technical report 00663790 (2012)
4. Killick, R., Fearnhead, P., Eckley, I. A.: Optimal detection of changepoints with a linear computational cost. arXiv:1101.1438, January 2011.
5. Marioni, J.-C., Thorne, N.-P., Tavare, S.: BioHMM: a heterogeneous hidden markov model for segmenting array CGH data. Bioinformatics **22**(9), 1144–1146 (2006)
6. Nicolas, P., Bize, L., Muri, F., Hoebeke, M., Rodolphe, F., Ehrlich, S.D., Prum, B., Bessieres, P.: Mining Bacillus subtilis chromosome heterogeneities using hidden Markov models. Nucleic Acids Res. **30**(6), 1418–1426 (2002)
7. Nicolas, P., Leduc, A., Robin, S., Rasmussen, S., Jarmer, H., Bessieres, P.: Transcriptional landscape estimation from tiling array data using a model of signal shift and drift. Bioinformatics **25**(18), 2341–2347 (2009)
8. Olshen, A.B., Venkatraman, E.S., Lucito, R., Wigler, M.: Circular binary segmentation for the analysis of array-based DNA copy number data. Biostatistics **5**(4), 557–572 (2004)
9. Picard, F., Lebarbier, E., Hoebeke, M., Rigaill, G., Thiam, B., Robin, S.: Joint segmentation, calling and normalization of multiple array CGH profiles. Biostatistics **12**(3), 413–428 (2011)
10. Picard, F., Robin, S., Lavielle, M., Vaisse, C., Daudin, J.-J.: A statistical approach for array CGH data analysis. BMC Bioinf. **6**, 27 (2005)
11. Picard, F., Robin, S., Lebarbier, E., Daudin, J.-J.: A segmentation/clustering model for the analysis of array CGH data. Biometrics **63**, 758–766 (2007)

12. Pique-Regi, R., Ortega, A., Asgharzadeh, S.: Joint estimation of copy number variation and reference intensities on multiple DNA arrays using GADA. Bioinformatics **25**(10), 1223–1230 (2009)
13. Rigaill, G.: Pruned dynamic programming for optimal multiple change-point detection. arxiv:1004.0887, April 2010
14. Shah, S.P.: Computational methods for identification of recurrent copy number alteration patterns by array CGH. Cytogenet. Genome Res. **123**(1–4), 343–351 (2008)
15. Teo, S.M., Pawitan, Y., Kumar, V., Thalamuthu, A., Seielstad, M., Chia, K.S., Salim, A.: Multi-platform segmentation for joint detection of copy number variants. Bioinformatics **27**(11), 1555–1561 (2011)
16. van de Wiel, M.A., Picard, F., van Wieringen, W.N., Ylstra, B.: Preprocessing and downstream analysis of microarray DNA copy number profiles. Brief. Bioinf. **12**(1), 10–21 (2011)
17. van de Wiel, M.A., Kim, K.I., Vosse, S.J., van Wieringen, W.N., Wilting, S.M., Ylstra, B.: CGHcall: calling aberrations for array cgh tumor profiles. Bioinformatics **23**(7), 892–894 (2007)
18. Willenbrock, H., Fridlyand, J.: A comparison study: applying segmentation to array CGH data for downstream analyses. Bioinformatics **21**(22), 4084–4091 (2005)
19. Zhang, N.R., Siegmund, D.O.: A modified Bayes information criterion with applications to the analysis of comparative genomic hybridization data. Biometrics **63**(1), 22–32 (2007)
20. Zhang, N.R., Siegmund, D.O., Ji, H., Li, J.Z.: Detecting simultaneous changepoints in multiple sequences. Biometrika **97**(3), 631–645 (2010)