Séminaire de Modélisation Statistique

On Optimal Multiple Changepoint Algorithms for Large Data

CRISTANCHO FAJARDO Lina
SORBA Marianne
ZLAOUI Khalil

Encadrement : LEBARBIER EMILIE

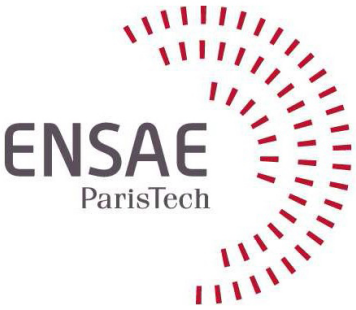

Table des matières

INTRODUCTION

Les données de signal sont souvent soumises à des changements soudains de structure, qu'il est nécessaire de prendre en compte pour une modélisation plus proche de la réalité. Les ruptures permettent de segmenter les données en plusieurs séquences, pouvant alors être modélisées séparément. De nombreux algorithmes de segmentation se sont donc développés à cette fin. Ils sont beaucoup utilisés en génétique, pour détecter la variabilité du nombre de copies d'un gène par exemple. Il y a de fait un intérêt pratique en cancérologie : les régions où le nombre de copies est supérieur ou inférieur à un seuil de référence peuvent indiquer une cellule cancéreuse. Le diagnostic du nombre de copies est donc crucial dans la détection et la classification du type de cancer.

L'article *On Optimal Multiple Changepoint Algorithms for Large Data* écrit par Robert Maidstone, Toby Hocking, Guillem Rigaill et Paul Fearnhead s'attache à expliciter les fondements de la segmentation, parcourant ainsi de façon linéaire les algorithmes existants, leurs applications, et leurs limites. Les auteurs se proposent d'améliorer ces algorithmes et de tester leur approche sur des problèmes génétiques classiques, notamment sur des données de micro-réseau de tumeurs. Cet article étant de nature très technique, et très spécifique à un domaine que nous découvrons, il paraît déplacé de formuler des critiques envers la démarche scientifique, ou de remettre en question les résultats énoncés. Nous choisissons vu le temps imparti à cet exercice, d'essayer de le retranscrire avec nos mots, et d'expliciter les principes et les applications des concepts énoncés. Afin de prendre plus de recul sur les techniques présentées, nous rapporterons dans un dernier temps des résultats d'articles scientifiques en lien avec le sujet qui nous ont paru pertinents.

1 Approche et développements

1.1 Approche

Une approche commune pour détecter les ruptures est de minimiser le coût sur chaque segmentation. Il s'agit ensuite de minimiser une version pénalisée de ce coût (problème de minimisation pénalisé), ou bien de minimiser le coût contraint par nombre de ruptures (problème de minimisation contraint). Il existe des méthodes de programmation dynamique (Auger et Lawrence, 1989 ou Jackson et al., 2005) afin de résoudre exactement ce problème de minimisation, mais couteux en temps de calcul (au moins quadratique en la longueur de la série à segmenter). Il existe aussi des algorithmes à l'instar de la segmentation binaire (Scott and Knott, 1974) dont le coût est quasi-linéaire en la longueur de la série, mais qui ne garantissent pas forcément de trouver la segmentation optimale. D'autres algorithmes sont entre deux (e.g. Olshen et al., 2004) et offrent des solutions plus précises pour un temps de calcul légèrement plus grand.

1.2 Développements

Les auteurs mentionnent le développement d'idées d'élagage relativement récentes, dans le but d'accélérer les méthodes dynamiques tout en résolvant le vrai minimum de la fonction de coût. Killick et al. (2012) présentent la technique d'élagage basé sur inégalité ("inequality based pruning"). Cette technique est la base de leur algorithme PELT, utilisé notamment pour résoudre des problèmes de minimisation pénalisé. Dans Rigaill (2010), une technique d'élagage fonctionnel ("functional pruning") est développée. Elle est la base de l'algorithme pDPA, utilisé pour résoudre un problème de minimisation contraint. Ces méthodes sont toutes deux optimales dans le problème qu'elles cherchent à résoudre. Les approches d'élagage sont cependant différentes, et d'une efficacité variable selon le jeu de données présenté. Ainsi, l'algorithme PELT est plus efficace lorsque le nombre de ruptures est grand, alors que l'algorithme pDPA l'est lorsqu'il y a peu de ruptures dans les données.

Cet article rappelle le fondement de ces méthodes, et cherche à les étendre avec notamment l'idée de les combiner. Les auteurs introduisent ainsi deux nouveaux algorithmes de segmentation : l'algorithme FPOP (avec élagage fonctionnel, "functional pruning") pour résoudre le problème de minimisation pénalisé et l'algorithme SNIP (avec élagage basé sur l'inégalité, "inequality based pruning") pour résoudre le problème de minimisation contraint. Leurs résultats montrent que le

FPOP élague toujours plus que son homologue PELT. Ils suggèrent aussi avec des résultats empiriques que l'algorithme FPOP est plus rapide que les méthodes de programmation dynamique actuelles, et que son efficacité est robuste au nombre de ruptures dans les données. Les données de micro-réseau des tumeurs s'étant imposées comme référence dans l'analyse de techniques de segmentation, à la fois en terme de précision et de vitesse de calcul, la méthode FPOP a ainsi été testée sur ce type de données. Les auteurs concluent que le FPOP a un coût de calcul comparable à celui de la segmentation binaire.

2 Modélisation

Le problème est modélisé par une série $(y_1, ..., y_i, ..., y_n)$ où i représente classiquement la position sur le chromosome dans un problème de nombre de copies. Nous reprenons donc les notations de l'article. Dans Figure 1 ci-dessous, τ_j ainsi représente la rupture $j = 1, ..., k$ et τ_0 la position de la première donnée y_0 (représentée par la première barre grise).

Le problème statistique tel que posé dans l'article est :

- Trouver k si celui-ci n'est pas fixé
- Trouver le lieu des ruptures, c-à-d τ_j pour $1 \leq j \leq k$

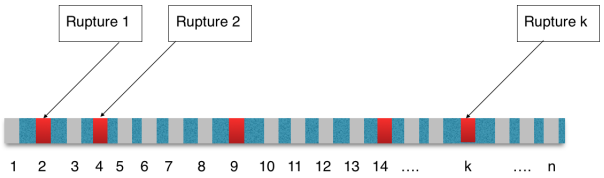

FIGURE 1 – Illustration de la segmentation

2.1 Méthode générique

Les auteurs expliquent que les méthodes de segmentation dépendent du type de changement que l'on souhaite détecter (changement en moyenne, en variance ou en distribution). En effet, il existe bien souvent un modèle préalable à la série. Pour illustrer plusieurs types de ruptures, nous empruntons une image extraite de la thèse écrite par Souhil Chaker (*Ségmentation de processus avec un bruit autorégressif*, 2015) :

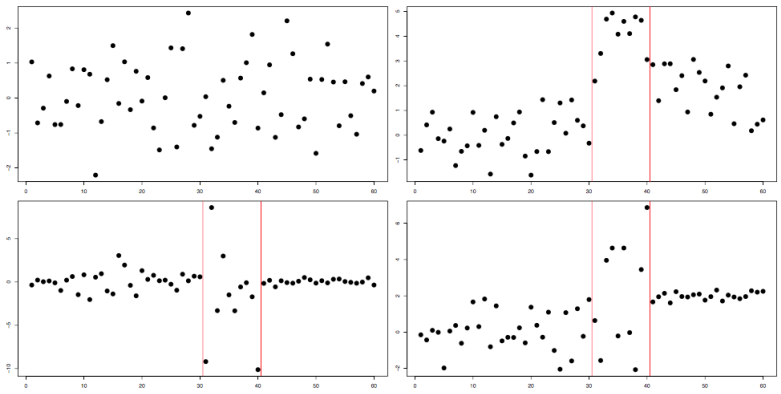

FIGURE 2 – Réalisation de 4 séries de 60 loi gaussiennes indépendants

Les droites verticales indiquent les ruptures éventuelles (localisées entre les données 30 et 40, ordonnées selon la position). En haut à gauche, il s'agit du cas i.i.d centré réduit. En haut à droite, d'une loi de variance constante égale à 1, avec ruptures dans l'espérance (égale à 0, puis à 4, puis à 2). En bas à gauche, l'espérance est constante égale à 0, avec des ruptures dans la variance (1, 16, puis 1/16). Enfin, en bas à droite, il y a des ruptures simultanées dans l'espérance et la variance avec ces mêmes chiffres.

Dans l'article qui nous concerne, les auteurs expliquent qu'il existe malgré la variabilité des problèmes à résoudre une structure générique dans la détection de ruptures qui consiste à :

- (i) Introduire une fonction de coût par segment
- (ii) Minimiser la somme des coûts $\sum_{j=0}^{k} \mathcal{C}(y_{\tau_j+1}, ..., y_{\tau_{j+1}})$ pour $1 \leq j \leq k$

Ainsi, si on connaît k, on peut en pratique inférer la position des ruptures en permutant la place des τ_j et en choisissant celle qui donne le coût minimum. Les auteurs indiquent que la forme de $\mathcal{C}$ dépend elle aussi du changement que l'on souhaite détecter, mais qu'il existe encore une fois une démarche générique :

- (iii) Modéliser le j-ème segment $[y_{\tau_j+1}, ..., y_{\tau_{j+1}}]$
- (iv) Maximiser la log vraisemblance sur chaque segment, ce qui revient à minimiser une fonction de coût égale à l'opposé de la log-vraisemblance : $-l_n(y_{\tau_j+1}, ..., y_{\tau_{j+1}}; \mu)$.
  N.B : Si les observations sont i.i.d, cette étape revient à faire :

$$\sum_{l=\tau_j+1}^{\tau_{j+1}} \min_{\mu}(-log(p(y_l|\mu))$$

Ainsi, la somme sur j (i.e. sur tous les segments), sera minimale :

$$\sum_{j=0}^{k} \sum_{l=\tau_j+1}^{\tau_{j+1}} \min_{\mu}(-log(p(y_l|\mu))$$

μ est un paramètre choisi judicieusement selon le type de changement à détecter, comme explicité ci-dessus. Comme dans l'étape (ii), on ne connaît pas k, il faut également l'estimer.

2.2 Méthode pratique

- (i) Pour estimer k, on définit :

$$C_{k,n} = \min_{\tau}(\sum_{j=0}^{k} \mathcal{C}((y_{\tau_j+1}, ..., y_{\tau_{j+1}}))) \tag{1}$$

- (ii) **cas contraint** : k est défini d'une façon indépendante. Fixer k et résoudre (1)
- (iii) Résolution avec un algorithme de programmation dynamique approprié, e.g. un algorithme de recherche de voisinage de segment ("Segment Neighbourhood Search algorithm")
- $(ii\ BIS)$ **cas pénalisé** : On ne connaît pas k. Fixer K de sorte que k varie dans $[1 : K]$. Le nombre de ruptures est estimé par la minimisation de : $C_{k,n} + f(k, n)$, où $f(k, n)$ est une fonction de pénalisation classiquement linéaire en k (par exemple, un AIC). On cherche de fait à résoudre :

$$\min_{k}[C_{k,n} + \beta k] = \min_{k,\tau}(\sum_{j=0}^{k} \mathcal{C}(y_{\tau_j+1}, ..., y_{\tau_{j+1}})) + \beta k$$

- $(iii\ BIS)$ Résolution avec un algorithme de programmation dynamique approprié, e.g. un algorithme de partitionnement optimal ("Optimal Partitioning algorithm")

2.3 Elagage

Les techniques d'élagage sont le centre d'intérêt de cet article, car elles sont le fondement de l'amélioration des algorithmes de programmation dynamique existants. Les auteurs rappellent ainsi qu'une des deux conditions suivantes doit être vérifiée pour l'application de cette technique :

- **Pour l'élagage fonctionnel** : Que pour une fonction γ :

$$\mathcal{C}((y_{\tau_j+1}, ..., y_{\tau_{j+1}})) = \min_{\mu}(\sum_{j=\tau_j+1}^{\tau_{j+1}} \gamma((y_i, \mu)) \tag{2}$$

- **Pour l'élagage basé sur inégalité** : qu'il existe κ tel que pour toute $t_j < s < t_{j+1}$:

$$\mathcal{C}((y_{\tau_j+1}, ..., y_S)) + C((y_{S+1}, ..., \tau_{j+1})) + \kappa \leq \mathcal{C}((y_{\tau_j+1}, ..., \tau_{j+1})) \tag{3}$$

3 Autour des algorithmes existants

Nous présentons dans cette partie les algorithmes de programmation dynamique existants tels qu'introduits par les auteurs. Ces algorithmes permettent de résoudre des problèmes de minimisation pénalisés et des problèmes minimisation contraints. Nous présentons par ailleurs les techniques d'élagage pouvant être utilisées dans l'objectif de réduire le coût de calcul des méthodes dynamiques existantes.

3.1 Solution existante au problème de minimisation contraint : Auger et Lawrence (1989)

L'idée de l'algorithme de Segment Neighbourhood Search (Auger et Lawrence (1989)) est la suivante : on cherche d'abord l'emplacement optimal de la dernière rupture τ_k. Après l'avoir trouvé, on réitère le procédé sur la série $(y_1, ..., y_i, ..., y_{\tau_k})$ que l'on souhaite séparer en k segments, afin de trouver l'emplacement de l'avant-dernière rupture. Le procédé s'arrête lorsque nous avons trouvé l'ensemble des k ruptures.
Notons $C_{l,t}$ le coût minimal de la segmentation de $(y_1, ..., y_i, ..., y_t)$ en $l+1$ segments. Il est démontré dans l'article la formule suivante :

$$C_{l,t} = \min_{\tau \in \{l,...,t-1\}} [C_{l-1,\tau} + \mathcal{C}(y_{\tau+1}, ..., y_t)]$$

Ainsi, l'emplacement optimal de la dernière rupture de la segmentation de $(y_1, ..., y_i, ..., y_n)$ en $k+1$ segments est le suivant :

$$\tau_k^*(n) = arg \min_{\tau \in \{k,...,n-1\}} [C_{k-1,\tau} + \mathcal{C}(y_{\tau+1}, ..., y_n)]$$

Pour trouver l'avant-dernière rupture $(k-1)$, il suffit d'effectuer le même calcul mais sur la segmentation $(y_1, ..., y_i, ..., y_{\tau_k^*(n)})$ que l'on sépare en k segments, et on réitère jusqu'à obtenir les k ruptures.
La complexité de cet algorithme est en $\mathcal{O}(n^2)$. En effet, pour tout $l \leq k$, il est nécessaire de calculer $C_{l,t}$ pour tout $t \in [\![1, n]\!]$. Pour calculer $C_{l,t}$, il faut calculer $C_{l-1,\tau} + \mathcal{C}(y_{\tau+1}, ..., y_t)$ pour tout $\tau \in [\![0, t-1]\!]$. Le nombre d'étapes est donc $k \sum_{t=1}^{n} t = k \frac{n(n+1)}{2} = \mathcal{O}(n^2)$

3.2 Élagage appliqué au problème de minimisation contraint : Cleynen et al. (2012)

Le principe de l'élagage repose sur le fait que sous certaines conditions, un certain nombre de points de la série ne pourront jamais être des futurs points de ruptures optimaux. Il est donc inutile de les considérer lors de la recherche de la segmentation optimale.
L'algorithme pDPA (pruned Dynamic Programming Algorithm) présenté dans l'article est basé sur la condition (2). Sous réserve que cette dernière est vérifiée, notons $Cost_{k,t}^{\tau}(\mu)$ le coût minimal de la segmentation de la série $(y_1, ..., y_i, ..., y_t)$ avec τ le dernier point de rupture et μ le paramètre du dernier segment. Notons $Cost_{k,t}^{*}(\mu) = \min_{\tau \leq t-1} Cost_{k,t}^{\tau}(\mu)$ et $C_{k,t} = \min_{\mu} Cost_{k,t}^{*}(\mu)$. Il est démontré dans l'article les formules suivantes :

$$\begin{cases} \forall \tau \leq t-1, Cost_{k,t}^{\tau}(\mu) = Cost_{k,t-1}^{\tau}(\mu) + \gamma(y_t, \mu) \\ \\ Cost_{k,t}^{*}(\mu) = \min\{Cost_{k,t-1}^{*}(\mu) + \gamma(y_t, \mu), C_{k-1,t}\} \end{cases}$$

Ces valeurs peuvent donc être actualisées à chaque étape sous réserve que μ soit un scalaire (afin que le minimum soit bien défini).
Le principe de l'algorithme est de définir $Set_{k,t}^{\tau} = \{\mu : Cost_{k,t}^{\tau}(\mu) = Cost_{k,t}^{*}\}$ comme l'ensemble des μ minimisant le coût de la segmentation en fixant τ comme l'emplacement de la dernière rupture. Il est démontré dans l'article la formule de récursion suivante :

$$Set_{k,t}^{\tau} = Set_{k,t-1}^{\tau} \cap \{\mu : Cost_{k,t}^{\tau}(\mu) \leq C_{k-1,t}\}$$

Ainsi, si $Set_{k,t}^{\tau} = \emptyset$, alors pour tout $T \geq t$, $Set_{k,T}^{\tau} = \emptyset$, c'est-à-dire que τ ne pourra plus jamais être l'emplacement optimal de la dernière rupture. Il est donc inutile de le considérer lors des

prochaines étapes de l'algorithme. De plus, si $Set_{k,t}^t = \emptyset$, t ne peut pas être un point de rupture : il n'est pas à considérer dans la recherche des points de ruptures, d'où le gain en temps. Ce gain n'est cependant pas systématique car l'algorithme nécessite le calcul et le stockage des fonctions $Cost_{k,t}^\tau(\mu)$ et des ensembles $Set_{k,t}^\tau$, qui peuvent être plus ou moins longs selon qu'ils nécessitent des méthodes analytiques ou numériques.

3.3 Solution existante au problème de minimisation pénalisé : Jackson et al. (2005)

Le problème de minimisation pénalisé est le suivant : Découper la série $(y_1, ..., y_i, ..., y_n)$ en $k+1$ segments distincts correspondant à k ruptures $(\tau_1, ..., \tau_i, ..., \tau_k)$. Contrairement au problème de minimisation contraint, le nombre de segments k n'est pas fixé au préalable. De façon à faire un compromis entre la précision du modèle et son coût de calcul, qui augmentent tous les deux avec k, il convient d'ajouter dans la fonction de coût une fonction qui pénalise la taille du modèle. Cette fonction f de pénalité est en général linéaire en la taille du modèle, avec $f(k, n) = \beta k$, $\beta > 0$. On cherche donc :

$$\min_k [C_{k,n} + \beta k] = \min_{k,\tau} [\sum_{j=0}^{k} \mathcal{C}(y_{\tau_j+1}, ..., y_{\tau_{j+1}})] + \beta k$$

Intéressons nous maintenant à l'algorithme de programmation dynamique existant présenté dans l'article (Jackson et al., 2005). L'idée est très proche de celle exposée dans le cas contraint : On construit un algorithme de récursion en trouvant d'abord l'emplacement optimal τ_n^* de la dernière rupture de la série $(y_1, ..., y_i, ..., y_n)$, puis en trouvant les points de ruptures de la série $(y_1, ..., y_i, ..., y_{\tau_n^*})$ qui minimisent le coût pénalisé de la segmentation de cette série.
Pour tout t, notons $F(t)$ la valeur minimale du coût pénalisé de la segmentation de la série $(y_1, ..., y_i, ..., y_t)$. Il est démontré dans l'article la formule récursive suivante :

$$F(t) = \min_{0 \leq \tau \leq t} [F(\tau) + \mathcal{C}(y_{\tau+1}, ..., y_t) + \beta]$$

Ainsi, l'emplacement optimal de la dernière rupture de la segmentation de $(y_1, ..., y_i, ..., y_t)$ est le suivant :

$$\tau_t^* = arg \min_{0 \leq \tau < t} [F(\tau) + \mathcal{C}(y_{\tau+1}, ..., y_t) + \beta]$$

Le vecteur cp(t) contenant tous les points de ruptures de la série $(y_1, ..., y_i, ..., y_t)$ s'obtient donc de manière récursive selon la formule suivante, avec $c(0) = \emptyset$:

$$cp(t) = (cp(\tau_t^*), \tau_t^*)$$

La complexité de cet algorithme est en $\mathcal{O}(n^2)$. En effet, pour calculer $cp(n)$, il est nécessaire de calculer $F(t)$ pour tout $t \in [\![1, n]\!]$. Pour calculer $F(t)$, il faut calculer $F(\tau) + \mathcal{C}(y_{\tau+1}, ..., y_t) + \beta$ pour tout $\tau \in [\![0, t-1]\!]$. Le nombre d'étapes est donc $\sum_{t=1}^{n} t = \frac{n(n+1)}{2} = \mathcal{O}(n^2)$

3.4 Élagage appliqué au problème de minimisation pénalisé : Killick et al. (2012)

L'algorithme d'élagage présenté dans l'article dans le cadre du problème de minimisation pénalisé est l'algorithme PELT ("Pruned Exact Linear Time"). Il repose sur la condition (3). Sous réserve que la condition (3) est vérifiée pour un κ, et si

$$F(t) + \mathcal{C}(y_{t+1}, ..., y_s) + \kappa > F(s)$$

il est montré que pour tout $T > s$, t ne pourra jamais être l'emplacement optimal de la dernière rupture de la segmentation de la série $(y_1, ..., y_i, ..., y_T)$. Le calcul de $F(t)$ est donc plus rapide parce que moins de points sont à considérer dans la minimisation :

$$F(t) = \min_{0 \leq \tau \leq t} [F(\tau) + \mathcal{C}(y_{\tau+1}, ..., y_t) + \beta] = \min_{\tau \in R_t} [F(\tau) + \mathcal{C}(y_{\tau+1}, ..., y_t) + \beta]$$

avec $R_{t+1} = \{\tau \in \{R_t \cup \{t\}\} : F(\tau) + \mathcal{C}(y_{\tau+1}, ..., y_t) + \kappa \leq F(t)\}$

4 Développement de nouveaux algorithmes

L'article présente deux nouveaux algorithmes d'élagage qui cherchent à étendre les algorithmes présentés ci-avant. Le premier est l'algorithme FPOP ("Functional Pruning Optimal Partitioning"). Il applique un **élagage fonctionnel** à l'algorithme de partition optimale de la section 3.1 dans le but d'améliorer son efficacité dans la résolution du problème pénalisé. Le deuxième algorithme introduit par les auteurs est l'algorithme SNIP ("Segment Neighbourhood with Inequality Pruning"), qui pour résoudre le problème contraint applique un élagage **basé sur inégalité** à l'algorithme de recherche de la section 3.2.

4.1 FPOP : Functional Pruning Optimal Partitioning

Comme le PELT, l'algorithme FPOP cherche à rendre plus efficace l'algorithme de partition optimale, mais cette fois-ci avec un élagage fonctionnel. Les auteurs montrent que le FPOP est plus efficace que le PELT dans certaines situations. L'approche de l'algorithme FPOP est similaire à celle du pDPA, notamment sur le fait que la condition (5) doit être vérifiée. Sa théorie est plus simple car le problème n'est plus contraint, et il n'y a donc pas besoin de conditionner par le nombre de ruptures.
Similairement à ce qui a été présenté dans la section 3.3, ils définissent :

$$Cost_t^\tau(\mu) = \begin{cases} F(\tau) + \sum_{i=\tau+1}^{t} \gamma(y_i, \mu) + \beta & \text{pour} \quad \tau \leq t-1 \\ F(\tau) + \beta & \text{pour} \quad \tau = t \end{cases}$$

qui correspond au coût minimal jusqu'au temps t, sous la condition que la dernière rupture soit au point τ, et que le dernier segment ait pour paramètre μ. Ils reprennent également la récursion suivante : $\tau \leq t-1 : Cost_t^\tau(\mu) = Cost_{t-1}^\tau(\mu) + \gamma(y_t, \mu)$. En suivant la même démarche que pour l'algorithme pDPA, mais en rendant les formules indépendantes du nombre de ruptures k, ils échangent l'ordre de la minimisation, pour étaler les valeurs de la dernière rupture potentielle τ, tout en permettant à μ de varier.

Ils définissent $Cost_t^*(\mu) = \min_\tau Cost_t^\tau(\mu)$, le coût minimal pour segmenter la série $(y_1, ..., y_i, ..., y_t)$, sous la condition que le paramètre du dernier segment soit μ.
Ainsi, $F(t) = \min_\mu Cost_t^*(\mu)$, ce qui permet de trouver une expression récursive :

$$\begin{aligned} Cost_t^*(\mu) &= min \left\{ \min_{\tau \leq t-1} Cost_t^\tau(\mu), Cost_t^t(\mu) \right\} \\ &= min \left\{ Cost_{t-1}^*(\mu), F(t) + \beta \right\} \end{aligned}$$

L'algorithme nécessite donc de stocker et d'actualiser efficacement les variables $Cost_t^*(\mu)$. Pour cela, et comme auparavant, les auteurs font une partition de l'espace D des valeurs possibles μ. Le problème revient ainsi à actualiser ces ensembles, et à stocker $Cost_t^\tau(\mu)$ pour chaque τ pour lequel l'ensemble est non vide. Ils définissent pour $\tau \leq t-1 : Set_t^\tau = \{\mu : Cost_t^\tau(\mu) = Cost_t^*(\mu)\}$ l'ensemble des μ minimisant le coût de la segmentation en fixant τ comme position la dernière rupture. D'où la récursion :

$$Set_t^\tau = Set_{t-1}^\tau \cap \{\mu : Cost_t^\tau(\mu) \leq F(t) + \beta\}$$

Ainsi, si $Set_t^\tau = \emptyset$ et $T > t : Set_T^\tau = \emptyset$, la valeur τ est objet d'élagage, c'est-à-dire, qu'elle n'est plus considérée dans les prochaines étapes de l'algorithme.
Pour t :

$$Set_t^t = D \setminus \left[\bigcup_\tau \cap \{\mu : Cost_t^\tau(\mu) \leq F(t) + \beta\} \right]$$

ils proposent de faire un élagage si $Set_t^t = \emptyset$. Finalement, comme dans l'algorithme PELT, ils posent R_t : l'ensemble des derniers points de rupture potentiels, et restreignent les règles d'actualisation pour $F(t)$ et τ_t^* de la section 3.3 à $\tau \in R_t$. L'ensemble R_t est actualisé récursivement à chaque étape par :

$$R_{t+1} = \{\tau \in \{R_t \cup \{t\}\} : Set_t^\tau \neq \emptyset\}$$

4.2 SNIP : Segment Neighbourhood with Inequality Pruning

L'élagage de l'algorithme SNIP est basé sur inégalité, contrairement au pDPA. L'algorithme nécessite donc la condition (6). Les auteurs montrent que $\forall k \geq 1$ et $t < s$:

$$C_{k-1,t} + \mathcal{C}(y_{t+1}, ..., y_s) + \kappa \leq C_{k-1,s}$$

et donc en tout temps futur $T > s$, que t ne peut pas être la position de la dernière rupture. Ils montrent que pour tout $s < T \leq n$, $C_{k,T} < C_{k-1,t} + \mathcal{C}(y_{t+1}, ..., y_T)$. t ne peut donc pas être la position optimale de la dernière rupture. Ceci implique que la règle d'actualisation sur $C_{l,t}$ présentée dans la section 3.1 peut être restreinte à un ensemble $R_{k,t}$:

$$R_{k,t+1} = \{v \in \{R_{k,t} \cup \{t\}\} : C_{k-1,v} + \mathcal{C}(y_{v+1}, ..., y_t) + \kappa \leq C_{k-1,t}\}$$

On retrouve une expression est d'ailleurs similaire à celle trouvée pour l'algorithme PELT.

4.3 Comparaison des algorithmes d'élagage

L'utilisation des méthodes d'élagage dépend des hypothèses que l'on peut formuler sur la fonction de coût. L'élagage fonctionnel nécessite une condition plus forte que celui basé sur inégalité. Ce dernier est plus coûteux en temps de calcul. Dans les cas où les deux peuvent être implémentés, les auteurs montrent que tout point objet d'élagage par la méthode basée sur inégalité (au temps t), le sera aussi par la méthode d'élagage fonctionnel. Ceci se justifie par le fait que la condition nécessaire pour l'élagage fonctionnel est plus forte. Les auteurs mettent en évidence le théorème empiriquement, en remarquant que dans le cas pénalisé, l'algorithme PELT élague très rarement, alors que le FPOP le fait plus fréquemment.

En ce qui concerne le cas contraint, l'algorithme pDPA fait plus souvent d'élagage que l'algorithme SNIP. Il demeure pourtant le problème du calcul, raison pour laquelle les auteurs testent la performance de FPOP par rapport à celle de PELT, du pDPA et de la ségmentation binaire (une référence en termes de vitesse de calcul). Les tests se font sur des données réelles et simulées. Dans le premier cas ils trouvent qu'en général l'algorithme FPOP est plus rapide que les algorithmes PELT et pDPA, mais qu'il l'est deux fois moins que l'algorithme de segmentation binaire. En utilisant les données simulées, les auteurs trouvent que la vitesse du PELT, de la segmentation binaire et du pDPA dépend du nombre de ruptures. Pour les deux derniers, si le nombre de ruptures est pus grand, il faut aussi augmenter K, le nombre maximum de ruptures . Quant au FPOP, ils confirment qu'il est plus efficace que les algorithmes PELT et pDPA : il est toujours plus rapide que ces algorithmes, et parfois même plus rapide que la segmentation binaire. Les auteurs concluent que le choix d'un algorithme dépend de l'application.

5 Pour aller plus loin

5.1 Détection de ruptures dans un signal avec dépendances

Cette discussion est basée sur l'article de Chakar, Lebarbier et al. (*A robust approach for estimating change-points in the mean of an AR(1) process*, 2015), et sur la thèse de Chakar (*Segmentation de processus avec un bruit autorégressif*, 2015). En effet les méthodes telles qu'elles ont été présentées dans cet article sont basées sur des algorithmes d'optimisation dynamique (voir section 2.2 , Méthode pratique). Chakar, Lebarbier et al. (2015) expliquent que dans le cas où il y a des dépendances dans les séries considérées, le calcul dynamique (seul algorithme donnant une solution optimale dans le cas indépendant), n'est plus valide. En effet, l'algorithme de programmation dynamique s'applique lorsque (i) la fonction de coût (la négative log-vraisemblance) est additive (respectivement aux segments), et lorsque (ii) aucun paramètre estimé ne peut être commun aux divers segments. Le Processus étudié est le suivant :

$$y_i = \mu_k^* + \eta_i, t_{n,k}^* \leq i \leq t_{n,k+1}^*, 0 \leq k \leq m^*, 1 \leq i \leq n$$

avec (η_i) un processus AR(1) Gaussien canonique ($|\rho^*| < 1$) défini comme solution de l'équation $(\eta_i) = \rho^* * (\eta_{i-1}) + (\epsilon_i)$, (ϵ_i) bruit blanc Gaussien de variance σ^{*2}.

Dans ce modèle, les paramètres ne changent pas forcément (et pas simultanément) à chaque rupture, et les conditions (i) et (ii) d'application de la programmation dynamique ne sont plus valides. En effet, la log-vraisemblance n'est pas additive respectivement aux segments à cause des dépendances qui existent avec les données de segments voisins, et le coefficient ρ^* doit donc être estimé de façon jointe sur tous les segments. Pour répondre à ce problème, Chakar, Lebarbier et al. proposent un estimateur robuste du paramètre d'auto-corrélation. Ceci leur permet non pas de connaître ρ^*, mais d'utiliser son estimation pour dé-corréler la série. Ils prouvent que l'estimateur proposé est robuste en la présence de ruptures (données considérées comme outliers). Ils montrent que les propriétés asymptotiques de l'estimateur sont les mêmes que dans les estimateurs classiques des modèles indépendants (traités dans l'article que nous commentons). Ils proposent ensuite de suivre l'approche inférentielle classique en injectant cet estimateur dans les critères utilisés pour l'estimation des ruptures.

5.2 Détection de ruptures avec contraintes d'orientation

Les algorithmes présentés dans l'article sont destinés à résoudre des problèmes sans contraintes entre les paramètres moyens de segments adjacents. Pourtant, dans *Optimizing ChIP-seq peak detectors using visual labels and supervised machine learningHocking*, Toby Dylan, et al.(2017), on apprend qu'il y a des applications où il est souhaitable d'imposer des contraintes sur les orientations des changements, c'est à dire entre les paramètres du modèle avant et après les ruptures, pour obtenir un modèle plus interprétable. Par example, pour les données génomiques de ChIP-seq, qui fournissent des mesures bruitées de la liaison ou de la modification des protéines dans l'ensemble du génome (*Practical Guidelines for the Comprehensive Analysis of ChIP-seq Data* Bailey et al., 2013), l'ajout d'une contrainte ascendante-descendante améliore la précision de détection de rupture, mais rend le problème d'optimisation plus compliqué.

Pour ce type d'application, l'algorithme le plus rapide est l'Algorithme de Programmation Dynamique Contraint (CDPA). Il présente cependant deux problèmes : il ne garanti pas de trouver la solution optimale, et est trop lent (avec une complexité de $O(Kn^2)$) pour être utilisée sur des données massives, comme c'est le cas pour beaucoup de bases de donnés génomiques.
L'article s'attache à montrer que dans ce cas, la technique d'élagage fonctionnel peut être appliquée. Ils introduisent l'algorithme GPDPA (Generalized Pruned Dynamic Programming Algorithm) qui fonctionne pour tout modèle de ruptures, tout en ayant la même complexité de temps log-linéaire $O(Kn\ logn)$ que l'algorithme pDPA. Ils remarquent que les algorithmes CDPA et GPDPA ont une performance comparable dans plusieurs ensembles de données ChIP-seq. En revanche, le pDPA sans contrainte se montre beaucoup moins performant, ce qui suggère que la contrainte est nécessaire dans le calcul d'un modèle de ruptures avec une précision optimale.

5.3 Détection de ruptures "de bas en haut"

Les algorithmes présentés dans l'article consistent à considérer les données comme un seul segment que l'on découpe progressivement à chaque itération : La segmentation se fait « de haut en bas ». L'article *Tail-greedy bottom-up data decompositions and fast multiple change-point detection* de Piotr Fryzlewicz (2017) propose une nouvelle approche qui consiste à effectuer la segmentation « de bas en haut ».

L'algorithme fusionne successivement les régions voisines des données qui sont le plus susceptibles de correspondre à un segment constant, plutôt que de subdiviser successivement les données les plus susceptibles d'être séparées par des points de ruptures. Cet algorithme, nommé « Tail-Greedy Unbalanced Haar » (TGUH), présente un certain nombre d'avantages par rapport aux autres approches :

- Il est efficace en pratique, rapide, et particulièrement adapté, notamment par rapport à l'algorithme PELT, à des signaux comportant un très grand nombre de points de ruptures.
- Par rapport aux méthodes "de haut en bas", l'algorithme TGUH se généralise plus facilement à des données plus complexes, comme des réseaux, des vidéos ou des images ou des données non structurées.
- L'algorithme peut faire en sorte, plutôt que de considérer les deux voisins les plus proches à chaque étape de fusion, de considérer des triplets ou encore des n-uplets. Ceci permettrait par exemple de détecter des points de ruptures dans un modèle polynomial par morceaux.

6 CONCLUSION

L'article étudié présente et développe deux nouveaux algorithmes d'élagage : l'algorithme FPOP, détectant les points de ruptures avec élagage fonctionnel dans le cadre du problème de minimisation pénalisée et l'algorithme SNIP, détectant les points de ruptures avec élagage basé sur une inégalité dans le cadre du problème de minimisation contraint. Ces deux algorithmes sont fortement inspirés et reposent sur les mêmes principes que les algorithmes existants pDPA et PELT et présentés au préalable par les auteurs.
Les quatre algorithmes se complètent mutuellement dans la mesure ou chaque algorithme est plus ou moins adapté à un certain type de données. Les avantages et inconvénients des algorithmes présentés ainsi que leurs caractéristiques peuvent être récapitulés dans la Figure 3 ci-dessous :

ALGORITHMES DE SEGMENTATION	
Objectif : minimisation du coût sur chaque ségmentation	
Problème de minimisation pénalisé	**Problème de minimisation contraint**
Consiste à minimiser une version pénalisée du coût	Consiste à minimiser le coût avec une contrainte sur le nombre de ruptures

SOLUTIONS GENERALES EXISTANTES AUX PROBLEMES DE MINIMISATION			
Programmation Dynamique		**Ségmentation Binaire**	
+	**-**	**+**	**-**
Solution exacte	Temps de calcul	Temps de calcul	Ségmentation pas toujours optimale

PROGRAMMATION DYNAMIQUE			
PROGRAMMATION DYNAMIQUE AVEC ELAGAGE (ALGORITHMES EXISTANTS)		PROGRAMMATION DYNAMIQUE AVEC ELAGAGE (NOUVEAUX ALGORITHMES PRESENTES DANS L'ARTICLE)	
Elagage basé sur inégalité "inequality based pruning"	Elagage fonctionnel "functional pruning"	Elagage fonctionnel "functional pruning"	Elagage basé sur inégalité "inequality based pruning"
PELT	**pDPA**	**FPOP**	**SNIP**
Pour résoudre un problème de minimisation pénalisé	**Pour résoudre un problème de minimisation contraint**	**Pour résoudre un problème de minimisation pénalisé**	**Pour résoudre un problème de minimisation contraint**
Efficace lorsque le nombre de ruptures dans les données est grand	Efficace lorsqu'il y a peu de ruptures dans les données	FPOP efficace pour des base de données volumineuses, peu importe le nombre de ruptures	

FIGURE 3 – Algorithmes de segmentation

Toutefois, ces quatre algorithmes de programmation dynamique restent similaires, et ne peuvent de fait être utilisés que dans un cadre restreint : Les données doivent être modélisables par un modèle appartenant à la famille exponentielle à paramètre univarié. Pourtant, la détection de ruptures est un domaine de recherche qui ne se limite ni à ce cadre, ni aux quatre méthodes dynamiques présentées dans l'article. Nous avons de fait tenté de rendre compte de la variété du domaine qu'est aujourd'hui la détection de ruptures en mentionnant plusieurs articles scientifiques traitant du problème des dépendances, des contraintes d'orientation et de la détection de "bas en haut"

BIBLIOGRAPHIE

- On optimal multiple changepoint algorithms for large data, Maidstone, R., Hocking, T., Rigaill, G. et al. (2016)
- Ségmentation de processus avec un bruit autorégressif, S. Chaker 2015
- Practical Guidelines for the Comprehensive Analysis of ChIP-seq Data, Bailey et al., 2013
- Tail-greedy bottom-up data decompositions and fast multiple change-point detection ,Piotr Fryzlewicz (2017)
- A robust approach for estimating change-points in the mean of an AR(1) process, 2015