IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Projet Capitrain

Séries temporelles

Enseignant-chercheur
Nicolas Beldiceanu

Elèves
Valentin Quiédeville
Vivien Louradour

27/11/2019

Introduction

Le but de ce projet est de développer un programme permettant de calculer différentes caractéristiques sur des séries temporelles.
Une série temporelle est décrite comme une suite d'entiers correspondant à des mesures effectuées à intervalles réguliers. Elles sont utilisées dans énormément de domaines, comme l'énergie (mesures d'électricité produite/consommée par exemple). Il est donc nécessaire d'avoir des outils permettant d'analyser ces séries temporelles, afin d'en dégager des caractéristiques (pique maximum par exemple).
Pour cela, nous nous sommes appuyés sur les travaux de Nicolas Beldiceanu, Mats Carlsson, Rémi Douence et Helmut Simonis dans l'article *Using Finite Transducers for Describing and Synthesising Structural Time-Series Constraints*. Cet article décrit formellement les contraintes applicables aux séries temporelles, et la manière de les calculer.

Choix d'implémentation

Nous avons décidé de développer le programme en Java pour plusieurs raisons :

- Rapidité d'exécution : Java est un langage compilé, qui s'exécute relativement rapidement, en comparaison avec d'autres langages comme le python par exemple qui est interprété.
- Portabilité : étant exécuté dans la JVM, le programme est compatible Linux, Windows et Mac.
- Notre connaissance : nous utilisons tous deux ce langage depuis longtemps et nous sommes donc à l'aise avec. De plus, Java est un langage connu par beaucoup, ce qui peut faciliter la reprise de notre code par quelqu'un d'autre.

Java étant un langage orienté objet, nous avons pu utiliser les mécanismes d'héritage. Cela à plusieurs avantages :

- Clarté du code
- Réutilisabilité : il est relativement facile de venir implémenter de nouvelles features ou agrégateurs grâce à l'abstraction
- Complexité cyclomatique du code: l'héritage nous permet de définir le pattern utilisé au lancement du programme, et donc d'éviter au maximum les conditions lors du calcul.

Au vu des résultats de performance donnés par ce programme (cf partie "passage à l'échelle"), et du manque de temps, nous n'avons pas implémenté de générateur de code.

En effet, du fait de l'utilisation massive de l'héritage, le code à une complexité cyclomatique très faible, et ne comprends pas (ou peu) de condition pendant le calcul (les seules conditions sont au début lors de l'instanciation de l'automate voulu).
Nous pensons donc que générer du code aurait été réellement bénéfique qu'en choisissant un langage de code généré plus performant que Java, comme du C, ce que nous n'avons pas eu le temps de faire.

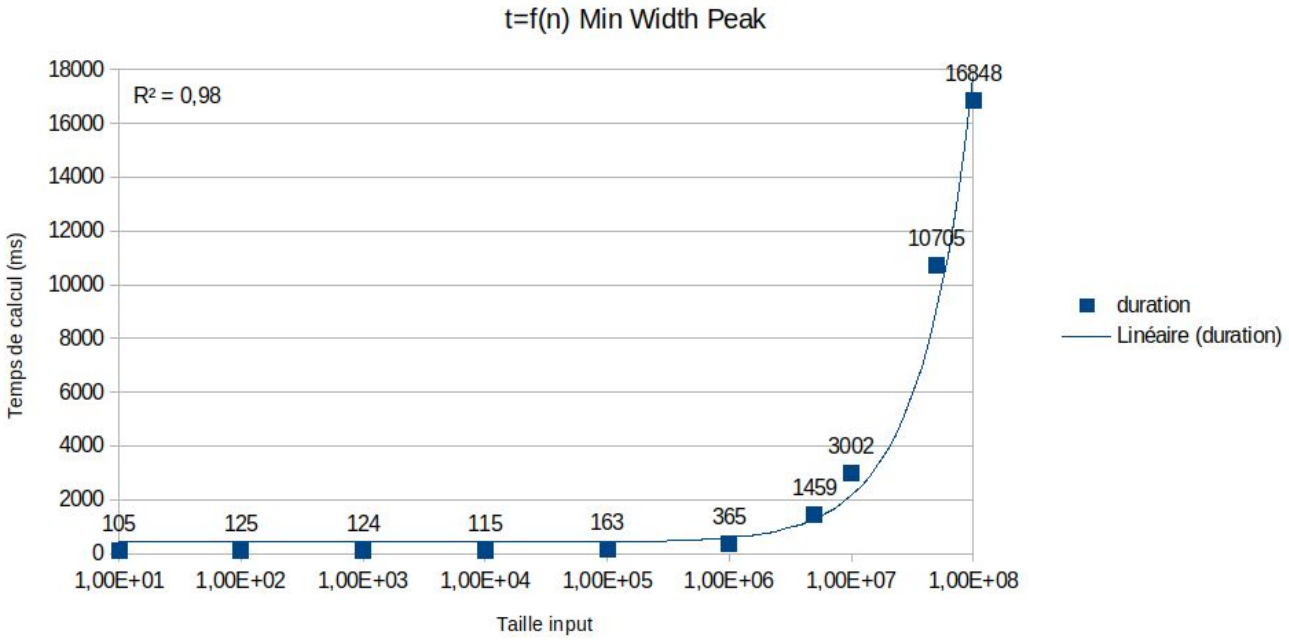
Validité des résultats

Pour valider nos résultats nous nous sommes appuyés sur les exemples du *Global Constraint Catalogue*. Nous avons ainsi rédigé des tests unitaires paramétrés sur les pattern suivant : *IncreasingSequence, IncreasingTerrace, Increasing, Peak, Plateau, ProperPlateau, StrictlyIncreasingSequence,Summit*.
Ces tests paramétrés utilisent les features Min, Max ,*Width*, et Surface, ainsi que les agrégateurs Min et Max.

Au total nous avons rédigé 42 tests unitaires couvrant des combinaisons de features, d'agrégateurs et de pattern avec des *before* et *after* de 0 et 1.

Passage à l’échelle

Afin d’effectuer des tests de passage à l’échelle, et de vérifier les performances de notre programme, nous avons décidé d’utiliser le chiffre pi.
En effet, cela permet d’avoir une suite d’entiers aussi longue que voulue, et sans répétition de pattern.
Voici les résultats obtenus sur l’algorithme Min Width Peak (échelle logarithmique) :

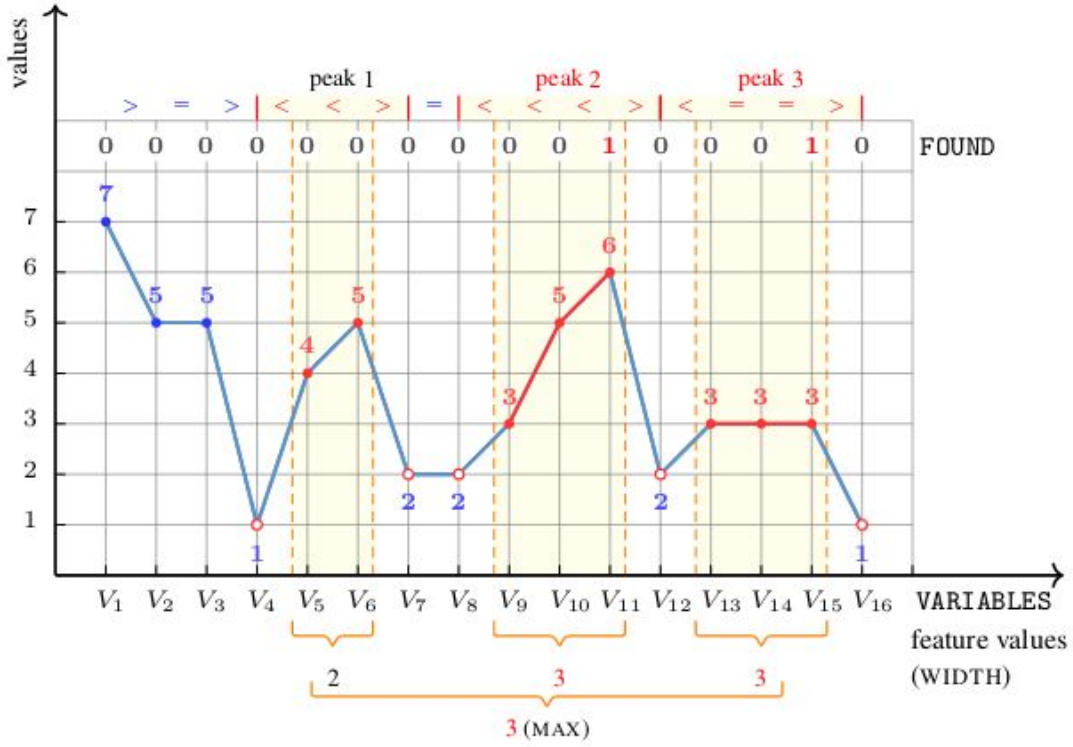


On voit donc que l’algorithme est en temps linéaire (coefficient R2 proche de 1).
L’algorithme prend moins d’une seconde jusqu’à plus de 1 million d’entiers (365ms pour 1 million), et environ 17s pour 100 millions d’entiers.
Par comparaison, l’étude des relevés de production d’électricité (environs 120.000 valeurs) RTE effectuée dans l’analyse de données (voir ci-dessous) a été faite en 150ms environ.

Les limites de notre implémentation

La consommation de mémoire de notre programme est importante. A titre d'exemple nous ne sommes pas parvenus à faire le calcul sur un fichier de 1Go contenant le premier milliard de décimal de PI. En effet, nous de disposons que de 10Go de mémoire vive au maximum ce qui n'est pas suffisant pour notre implémentation.

La seconde limite de notre implémentation réside dans le fait que nous ne sommes pas parvenus dans le temps imparti à implémenter les *guards*. Cependant, nous sommes capables de déterminer la position du pattern respectant l'agrégateur et la feature.
Illustrons notre propos avec l'exemple suivant :



Si nous utilisant cette série temporelle avec notre implémentation en utilisant le pattern *peak* l'agrégateur MAX, et la *feature* WIDTH, notre programme renvoie les données suivantes :

- MAX_WIDTH = 3
- iOccurences = {[8,10],[12,14]}

Analyse de données

Afin de tester notre programme sur des données pertinentes, nous avons utilisé les jeux de données fournis par RTE (Réseau de Transport d'Electricité).
Nous avons utilisé les relevés de production (toute filière confondue) et de consommation d'électricité en France entre 2013 et aujourd'hui.

Nous avons utilisé la feature et l'agrégateur max sur le pattern peak, afin de détecter les plus gros pics de consommation. Nous nous attendions à avoir un pic de production/consommation au même moment, mais ce ne fût pas le cas :

- Pic de production :
    - Sortie console :

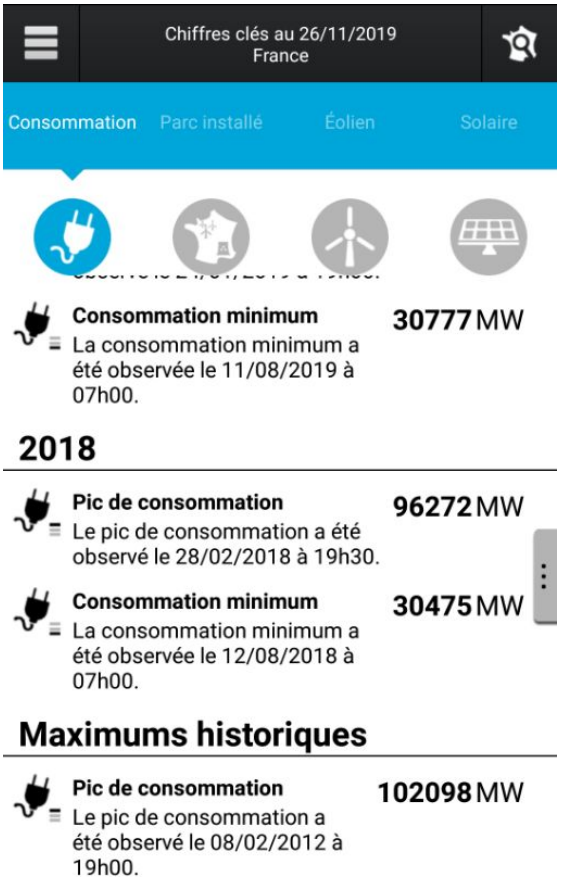
```
Starting with: MAX MAX ./resources/pattern/peak.csv
Data file: ./resources/input/other_dataset/production-quotidienne-totale.digt
118272
Translation time : 106 ms
Automaton Time : 51 ms
Total Time : 157 ms
1 matches
Value : 94233, startXi : 36791, endXi : 36799
```

    - Interprétation :
      Le pic de production était de 94 233 MW, entre le 06/02/2015 à 11h00 et le 06/02/2015 à 14h30.
- Pic de consommation :
    - Sortie console :

```
Starting with: MAX MAX ./resources/pattern/peak.csv
Data file: ./resources/input/other_dataset/consommation-quotidienne-totale.digt
118272
Translation time : 116 ms
Automaton Time : 43 ms
Total Time : 159 ms
1 matches
Value : 96272, startXi : 90467, endXi : 90475
```

    - Interprétation :
      Le pic de consommation était de 96 272 MW, entre le 28/02/2018 à 17h00 et le 28/02/2018 à 21h00.

Nous avons pu valider l'information sur le pic de consommation grâce à l'application mobile RTE qui propose une visualisation de leurs données :



On voit bien que le pic de consommation à été observé dans l'intervalle de temps indiqué par notre programme (celui-ci ne retournant qu'un intervalle et non une position exacte), et que la valeur de ce pic est bien exact.