Apprentissage automatique d'automates temporisés à partir de séries temporelles

Lénaïg Cornanguer

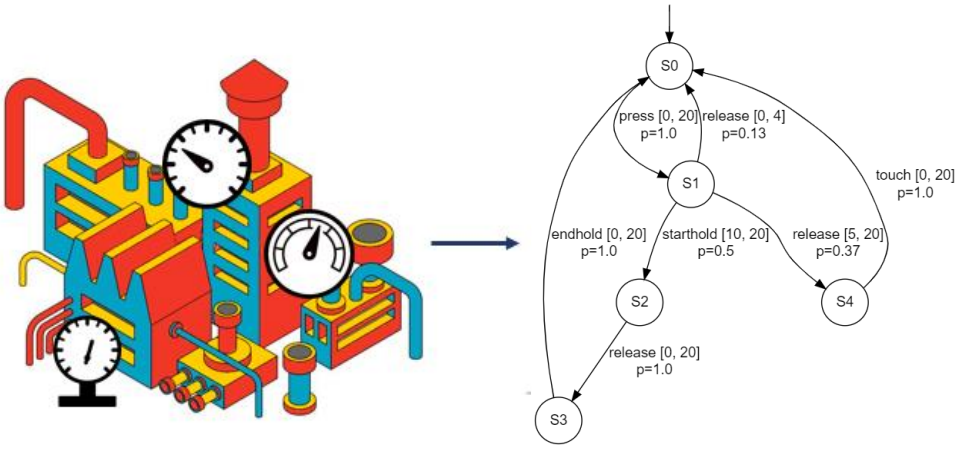
AGROCAMPUS OUEST

☐ **CFR Angers** ☑ **CFR Rennes**

Année universitaire : 2019-2020 Spécialité : Ingénieur Agroalimentaire Spécialisation (et option éventuelle) : Science des données	**Mémoire de fin d'études** ☑ d'ingénieur de l'École nationale supérieure des sciences agronomiques, agroalimentaires, horticoles et du paysage (AGROCAMPUS OUEST), école interne de l'institut national d'enseignement supérieur pour l'agriculture, l'alimentation et l'environnement ☐ de master de l'École nationale supérieure des sciences agronomiques, agroalimentaires, horticoles et du paysage (AGROCAMPUS OUEST), école interne de l'institut national d'enseignement supérieur pour l'agriculture, l'alimentation et l'environnement ☐ d'un autre établissement (étudiant arrivé en M2)

Apprentissage automatique d'automates temporisés à partir de séries temporelles

Par : Lénaïg CORNANGUER



Soutenu à Rennes le 8 septembre 2020

Devant le jury composé de :

Président : David Causeur

Maître de stage : Christine Largouët et Laurence Roze

Enseignant référent : Mathieu Emily

Les analyses et les conclusions de ce travail d'étudiant n'engagent que la responsabilité de son auteur et non celle d'AGROCAMPUS OUEST

Remerciements

Tout d'abord, je souhaite remercier Christine Largouët pour m'avoir fait confiance pour ce sujet de stage très orienté informatique.

Je souhaite également remercier l'intégralité des professeurs de la spécialisation Sciences des données pour les connaissances qu'ils nous ont apportées, et tout particulièrement Mathieu Emily, mon enseignant référent, qui a pris le temps de répondre à mes interrogations durant ce stage.

Je remercie Laurence Roze et une nouvelle fois Christine pour leur temps, leurs conseils et leurs encouragements tout au long de ce stage.

Je remercie Paul pour nos discussions sur les applications possibles en industrie et pour ses conseils de rédaction.

Je remercie Willy pour son aide précieuse durant la phase de relecture de ce mémoire.

Enfin, j'aimerais remercier l'ensemble de l'équipe LACODAM pour leur accueil chaleureux et je me réjouis de pouvoir passer trois années supplémentaires à leurs côtés.

Table des matières

Liste des abréviations

ABBA Adaptive Brownian Bridge-based symbolic Aggregation. 4, 6, 7, 14

APTA Augmented Prefix Tree Acceptor. 2, 4, 7–9

AT Automate Temporisé. 1–4, 7, 11–14, 16, 19, 20

DBI Davies-Bouldin Index. 5, 6, 15

DRTA Deterministic Real-Time Automaton. 4, 11

EDSM Evidence-Driven State-Merging. 2

EFD Equal Frequency Discretization. 4

EM Expectation-Maximization. 6

EWD Equal Width Discretization. 4

KDE Kernel Density Estimation. 4, 5

MMC Modèle de Markov Caché. 4, 6, 15–18, 20

NEP Nettoyage En Place. 12, 13, 17

PAA Piecewise Aggregate Approximation. 4, 5, 14

PDRTA Probabilistic Deterministic Real-Time Automaton. 2, 4, 7, 16

SAX Symbolic Aggregate approXimation. 4, 5, 14, 15

ST série(s) temporelle(s). 1, 2, 4–7, 11–15, 19, 20

Liste des symboles

μ Moyenne. 13

% Pourcent. 13

σ^2 Variance. 13

Liste des figures

Liste des tableaux

Introduction

Un automate est une méthode formelle utilisée pour modéliser des systèmes à états, où la transition d'un état à un autre est conditionnée par des actions ou événements. Ces systèmes ont généralement un nombre fini d'états possibles, on parle alors d'automate fini. La représentation commune d'un automate fini est un diagramme états-transitions sur lequel on va pouvoir identifier l'état initial du système et les états finaux à savoir les états sur lesquels le système peut terminer un cycle de fonctionnement. Dans la plupart des systèmes, la dimension temporelle est prépondérante et les actions sont réalisées à des temps spécifiques. Par exemple, une action ne pourra pas être effectuée avant un délai donné avec l'action précédente. Afin d'inclure cette dimension temporelle à la théorie des automates, Alur and Dill ont proposé en 1994 la notion d'Automate Temporisé (AT). En plus d'être constitués d'un ensemble fini d'états pouvant être initiaux et/ou finaux et de transitions étiquetées par les actions les conditionnant, ils sont associés à un ensemble fini de variables temporelles nommées horloges, qui permettent de définir des contraintes temporelles. Ces horloges sont des variables s'incrémentant de manière continue à chaque pas de temps défini. Elles peuvent être réinitialisées au cours du passage d'une transition et leur valeur peut conditionner la possibilité de passer d'un état à un autre. Les AT sont aujourd'hui utilisés dans de nombreux domaines. Par exemple, ils sont utilisés pour modéliser des écosystèmes et explorer différents scénarios sur leur évolution (Largouët et al., 2012). Ils sont aussi utilisés pour modéliser des protocoles de sécurité sur un réseau informatique afin d'en vérifier la justesse et d'identifier des comportements anormaux (Kurkowski et al., 2013). En effet, la force de ces modèles réside dans les outils développés pour cette syntaxe, permettant de vérifier des propriétés de système exprimées en logique mathématique avec le model checking, ou de générer des scénarios de test avec la synthèse de contrôleur. Cependant, les AT sont actuellement produits manuellement à partir des connaissances que les experts peuvent avoir du système. Cette méthode de construction induit un risque d'erreurs, est chronophage et ne permet pas de modéliser des systèmes dont on ne connaît pas le fonctionnement.

Dans un même temps, la multiplication des capteurs sur les systèmes physiques collectant des séries temporelles ST et de l'enregistrement automatique des logs de processus (historique horodaté des événements) permet l'accès à des données liées au fonctionnement des systèmes. Ces données sont généralement exploitées par des techniques de machine learning ou de data mining. Le modèle créé sert le plus souvent à résoudre une tâche bien précise telle que la prédiction ou encore la détection d'anomalies.

Le travail présenté dans ce mémoire concerne l'utilisation de ces données pour la modélisation des systèmes dont elles sont issues sous forme d'AT, ce qui permettrait de modéliser des systèmes dont on ne maîtrise pas le fonctionnement. L'intérêt est d'apprendre un modèle complet permettant l'usage de techniques de modèle checking ou de synthèse de contrôleur et donc d'étendre les tâches issues de l'apprentissage afin de vérifier, de comprendre ou de contrôler le système. La problématique est alors la recherche d'une méthode d'apprentissage automatique d'AT à partir de ST.

Pour ce faire, la première partie du mémoire est constituée par l'état de l'art des techniques actuelles d'apprentissage automatique d'AT ainsi que des méthodes de discrétisation de ST. La seconde partie du mémoire présente un nouvel algorithme d'apprentissage automatique d'AT. La troisième partie présente les expérimentations réalisées avec cet algorithme. Enfin, la dernière partie est une discussion des travaux présentés dans ce mémoire.

1 Etat de l'art

Nous verrons d'abord les méthodes existantes d'apprentissage automatique d'AT. Ensuite, différentes méthodes de discrétisation permettant de faire le lien entre ST et apprentissage d'AT seront présentées.

1.1 Apprentissage automatique d'AT

Formellement, un AT A est un 6-uplet $A = (Q, \Sigma, C, E, q_0, F)$ où Q est un set fini d'états, Σ est un set fini d'actions, C est un ensemble fini d'horloges, E est un ensemble fini de transitions, $q_0 \in Q$ est l'état initial et $F \subseteq Q$ est un set d'états finaux.

Une transition $e \in E$ est un 5-uplet $e = (q, a, g, r, q')$ où q et q' sont respectivement l'état source et l'état de destination, g est une garde (contrainte sur la valeur d'une horloge), et r le set d'horloge étant réinitialisées lors de son emprunt.

L'apprentissage automatique d'AT est une problématique récente et démontrée comme étant NP-difficile (Verwer et al., 2012). De part cette complexité, les approches actuelles se concentrent sur l'apprentissage de sous-classes d'AT ou utilisent des algorithmes d'approximation. Trois approches sont présentes dans la littérature. Les deux premières sont des adaptations de méthodes utilisées pour apprendre des automates non temporisés, tandis que la troisième approche est basée sur la programmation génétique.

RTI+ (Verwer et al., 2010) est un algorithme permettant d'apprendre une sous classe d'AT, le Probabilistic Deterministic Real-Time Automaton (PDRTA). Un PDRTA est un automate temporisé où les gardes sur les horloges sont réduites à des intervalles de délai avec l'événement précédent. La notion de déterminisme signifie qu'au niveau de chaque état, il ne peut exister plus d'une transition sortante pour un symbole (événement) donné et un délai donné. Enfin, le PDRTA est dit probabiliste car pour chaque transition est calculée une probabilité d'emprunt de celle-ci au cours du temps. La figure 1 présente un PDRTA. Les données d'apprentissage prennent la forme de séquences de symboles

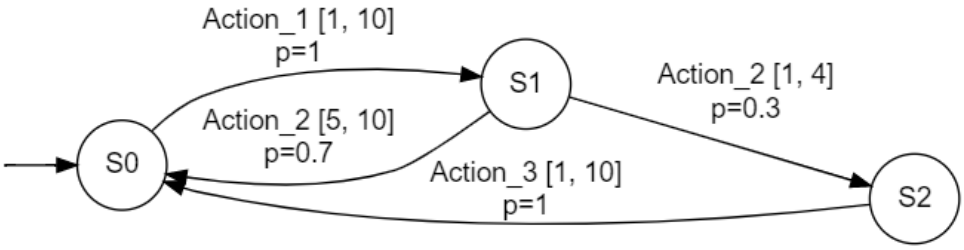


FIGURE 1 – Un PDRTA. Chaque transition est étiquetée par un symbole (Action_1, Action_2, ...) une garde ([1, 10], [1, 4], ...) et une probabilité d'emprunt en partant de l'état source.

avec le délai écoulé entre chaque réalisation d'événement. RTI+ est basé sur l'algorithme Evidence-Driven State-Merging (EDSM), très utilisé en inférence grammaticale pour l'apprentissage d'automates non temporisés (Lang et al., 1998). Tout d'abord, un automate en forme d'arbre nommé Augmented Prefix Tree Acceptor (APTA) est construit à partir du jeu de données d'apprentissage. Un APTA est un automate où il existe exactement une seule trajectoire pour atteindre un état donné. Sur chaque transition de l'APTA, la garde est fixée aux extrema de délais observés dans l'ensemble du jeu de données. Les paramètres probabilistes sont fixés sur les fréquences observées dans les données. L'APTA est ensuite transformé de manière ascendante en partant de l'état initial, avec deux opérations de transformation possibles : la division de transition et la fusion d'état. La division d'une

transition consiste en sa duplication avec un partage de la garde de sorte que le choix reste déterministe. La fusion de deux états consiste en l'accumulation des transitions entrantes et sortantes sur ces derniers sur un seul et la suppression du second état. Si cette fusion entraîne un choix non déterministe, les états de destination des transitions posant problème sont aussi fusionnés, avec répétition de cette étape jusqu'à ce que l'automate soit de nouveau déterministe. C'est le processus de déterminisation. Le choix de l'opération et le lieu est basé sur l'évidence, à savoir que le choix se portera sur la transformation permettant la plus grande augmentation de la vraisemblance. Cette vraisemblance est calculée grâce aux paramètres probabilistes.

Timed k-Tail (Pastore et al., 2017) est un algorithme permettant d'apprendre un AT à partir de traces constituées des événements qui ont été exécutés et de leur durée. Il est basé sur l'algorithme k-Tail (Biermann and Feldman, 1972), développé pour l'apprentissage d'automates non temporisés. Timed k-Tail est prévu pour modéliser des processus avec des événements imbriqués. Pour cette raison, pour un événement donné, il y aura deux transitions (une pour son début et une pour son arrêt). L'AT appris par Timed k-Tail possède deux types d'horloges. Tout d'abord, une horloge globale non réinitialisable mesure le temps écoulé depuis le début de la trajectoire. Il existe ensuite une horloge pour chaque événement, réinitialisée lors de son apparition et vérifiée au niveau de sa valeur sur la transition de fin de l'événement. Au niveau de chaque état de l'automate appris par Timed k-Tail, le choix de transition n'est déterministe que si l'on connaît les événements qui suivront (le nombre de transitions à regarder est déterminé par un paramètre k fixé à 2 dans la version distribuée de l'algorithme). La figure 2 présente le même automate que précédemment dans la syntaxe utilisée par Timed k-Tail. Au démarrage de l'algorithme,

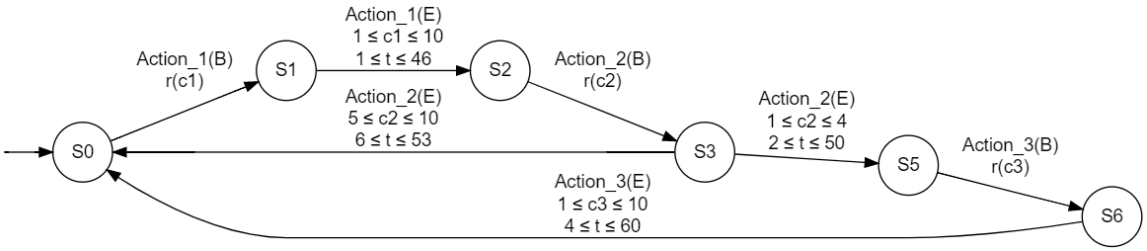


FIGURE 2 – Un AT avec la syntaxe de Timed k-Tail. Les transitions correspondant au début d'un événement ont leur symbole accolé d'un « (B) » et une horloge est réinitialisée (r(c1), r(c2)...). Les transitions correspondant à la fin d'un événement ont leur symbole accolé d'un « (E) », une vérification de la valeur de l'horloge de l'événement et une vérification de l'horloge globale t.

un automate en forme d'arbre où chaque branche correspond à une trace du jeu de donnés est construit. Pour chaque état de cet automate, est calculé le set de trajectoires possibles constituées de k transitions en partant de cet état. C'est le « futur » de l'état. Ensuite, les états ayant les mêmes futurs sont fusionnés dans une démarche ascendante en partant de l'état initial. Lors des fusions d'états, les contraintes sur les horloges sont accumulées. Par la suite, les horloges réinitialisées et vérifiées sur les mêmes transitions seront fusionnées.

GenProgTA (Tappler et al., 2018) est un algorithme apprenant des AT en se basant sur la programmation génétique. L'AT appris est déterministe et peut posséder plusieurs horloges. La figure 3 présente le même automate que précédemment dans la syntaxe utilisée par GenProgTA. L'algorithme commence par la génération de deux populations d'AT de manière aléatoire. La première population, la population dite globale, va être évaluée avec l'ensemble du jeu de données d'apprentissage. La seconde population, la population

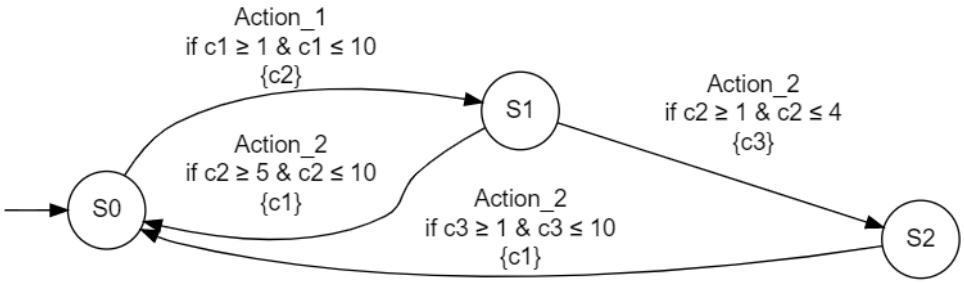


FIGURE 3 – Un AT avec la syntaxe de GenProgTA. Les transitions sont étiquetées par un symbole, la vérification de valeurs d'horloges et des réinitialisations de ces dernières entre accolade.

dite locale, va être évaluée par les séquences qui ont été rejetées par le meilleur automate de la population globale. La notion de « meilleur automate » est permise par un score calculé pour chaque automate à partir de sa taille (nombre d'états), du nombre de séquences qu'il accepte, et de son déterminisme. Trois opérations peuvent alors être réalisées : une mutation (division de transition, ajout d'une réinitialisation d'horloge, ajout d'un état...), un cross-over (échange d'une partie des automates) entre deux AT d'une même population, ou un cross-over entre un AT de la population globale et un de la population locale. L'opération est choisie de manière aléatoire. Seuls les meilleurs automates sont conservés et ajoutés aux nouvelles populations alors générées. L'algorithme s'arrête lorsqu'un automate accepte toutes les séquences du jeu de données d'apprentissage.

Tous ces algorithmes utilisent des traces horodatées de symboles déjà existants. Si l'on souhaite exploiter des ST pour l'apprentissage d'un AT, il est donc nécessaire de passer par une phase de discrétisation pour obtenir ces symboles.

1.2 Discrétisation de séries temporelles

La discrétisation de ST ou symbolisation est la transformation de ST en séquences de symboles contenant la part la plus importante possible de l'information temporelle et quantitative initialement présente. Ce processus permet l'augmentation du rapport signal-bruit et la réduction de la dimensionnalité, et donc une exploration des données plus efficace. Les méthodes de symbolisation peuvent être statiques ou dynamiques. Les premières ne prennent pas en compte la dimension temporelle des données tandis que les secondes considèrent à la fois les valeurs et leur succession.

1.2.1 Méthodes statiques

Les méthodes statiques partitionnent l'espace des valeurs mesurées et associent chaque intervalle à un symbole. Les stratégies les plus simples choisissent des points de ruptures créant des classes équiprobables ou en fixant une largeur de d'intervalle. Ces méthodes sont respectivement nommées Equal Frequency Discretization (EFD) and Equal Width Discretization (EWD) (Zalewski et al., 2012).

La Kernel Density Estimation (KDE) peut aussi être utilisée en tant que méthode statique de symbolisation (Biba et al., 2007). Un noyau choisi par l'utilisateur est associé à chaque point de la ST. Un paramètre nommé largeur de bande contrôle la taille de ces noyaux. Les noyaux individuels sont ensuite sommés sur l'espace des valeurs pour obtenir une estimation de densité globale. La largeur de bande contrôle le lissage du résultat. L'espace des valeurs est finalement partitionné au niveau des minima locaux (Figure 4).

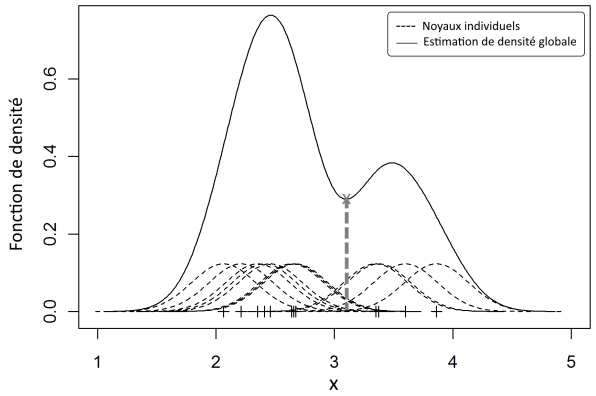


FIGURE 4 – Identification des points de rupture avec la méthode KDE.

Le clustering par les k-Means peut être adapté pour partitionner des ST sans considérer la dimension temporelle. Tandis que le partitionnement de données en k clusters pour tout cas en dimension $d \geq 2$ est NP-difficile, il est possible d'identifier une partition optimale en dimension $d = 1$ (Grønlund et al., 2018). L'algorithme identifie itérativement les k centroïdes minimisant l'inertie, la somme de la distance euclidienne entre chaque point d'un cluster et le point moyen (centroïde) de ce cluster. Le nombre de cluster k peut être sélectionné en calculant le silhouette score ou le Davies-Bouldin Index (DBI).

1.2.2 Méthodes dynamiques

Les méthodes dynamiques peuvent être divisées en deux catégories. Certaines méthodes vont analyser la dynamique temporelle mais le partitionnement sera effectué sur l'espace des valeurs. D'autres méthodes vont directement partitionner l'espace temporel.

Symbolic Aggregate approXimation (SAX) (Lin et al., 2003) est l'une des méthodes les plus connues pour discrétiser des ST. Tout d'abord, la dimension des données est réduite en appliquant une Piecewise Aggregate Approximation (PAA). Les données sont divisées en w classes de taille égale. Dans la représentation PAA, chaque point d'une classe prend la valeur de sa moyenne. Sous l'hypothèse que les ST centrées et réduites ont une distribution normale (Larsen and Marx, 1986), les points de rupture sont les points produisant le nombre désiré d'aires égales sous la courbe gaussienne (Figure 5). SAX considère la dimension temporelle des données uniquement durant la PAA et par l'utilisation de cette hypothèse.

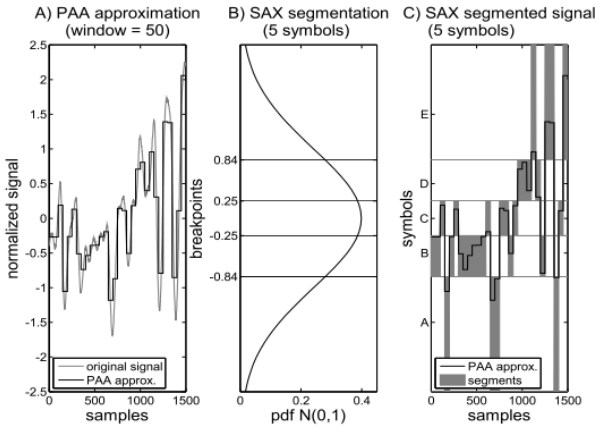
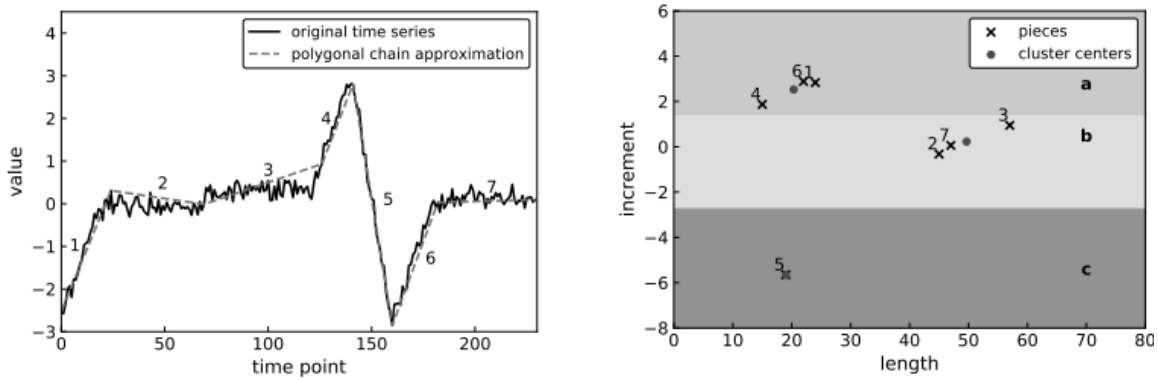


FIGURE 5 – PAA et segmentation par SAX. (Sant'Anna and Wickstrom, 2011)

Les ST peuvent être discrétisées en modélisant un Modèle de Markov Caché (MMC) sous l'hypothèse que le système étant modélisé est un process de Markov avec des états cachés, un modèle de Markov de premier ordre et homogène dans le temps. À chaque pas de temps, l'état caché courant ne dépend que de l'état caché au pas de temps précédent. En donnant un nombre supposé d'états cachés, l'algorithme forward-backward (ou algorithme de Baum-Welch) permet d'estimer les paramètres du modèle à partir des observations. C'est un algorithme de type Expectation-Maximization (EM). Pour ce faire, il part d'un modèle initial pouvant être choisi aléatoirement puis calcule la probabilité de la séquence d'états cachés sachant le modèle et les observations. C'est la phase d'évaluation de l'espérance de la vraisemblance (étape E). Ensuite, il estime le maximum de vraisemblance des paramètres en utilisant la vraisemblance trouvée à l'étape E. C'est la phase de maximisation de la vraisemblance (étape M) qui permet d'obtenir les paramètres actualisés du modèle. En itérant sur ces deux étapes, il parvient finalement à une estimation des paramètres du modèle qui maximise localement la probabilité de la séquence observée. Avec les paramètres obtenus, l'algorithme de Viterbi va ensuite être utilisé pour découvrir la meilleure séquence d'états cachés pour les données en simplifiant le calcul par programmation dynamique. Cette séquence d'états correspond à la ST initiale discrétisée.

Persist (Mörchen and Ultsch, 2005) est une méthode de discrétisation de ST partitionnant l'espace des valeurs en optimisant la persistance des symboles au cours du temps. Sous l'hypothèse d'un modèle de Markov de premier ordre, s'il n'y a pas de structure temporelle dans les données, les symboles peuvent être interprétés comme des observations indépendantes d'une variable aléatoire. La probabilité d'auto-transition (répétition d'un même symbole) est alors égale à la probabilité marginale d'apparition de ce symbole. En cas de comportement de persistance, la probabilité d'observer une auto-transition est supérieure à la probabilité marginale de ce symbole. La divergence symétrique de Kullback-Leiber est utilisée pour comparer la probabilité marginale et la probabilité d'auto-transition d'un symbole. Un score de persistance résume cette information pour l'ensemble des symboles. L'algorithme commence par déterminer 100 classes à fréquence égale soit 99 points de rupture potentiels. Pour chaque point de rupture potentiel, le score de persistance est calculé. Le point de rupture donnant le meilleur score est sélectionné. Cette étape est réalisée jusqu'à l'obtention du nombre de symboles désiré. L'algorithme s'arrête lorsqu'il n'y a pas d'amélioration possible du score de persistance par l'ajout d'un point de rupture.

Adaptive Brownian Bridge-based symbolic Aggregation (ABBA) (Elsworth and Güttel, 2020) est une méthode de symbolisation de ST en deux phases : une phase de réduction de dimension des données et une phase de clustering. ABBA cherche à identifier des motifs se répétant et caractérisés par leur longueur et leur valeur d'incrémentation. La dimension est réduite par l'approximation de la ST en une fonction linéaire, continue et régulière par morceaux. Il en résulte une séquence de 2-uplets constitués de la longueur du segment et de sa valeur d'incrémentation (Figure 6a). Ces tuples sont ensuite partitionnés par la méthode des k-Means (Figure 6b) où il est possible de pondérer l'importance de la longueur respectivement à celle de la valeur incrémentale. Le nombre de clusters est identifié grâce au DBI ou peut être fixé par l'utilisateur.



(a) Phase de réduction de dimension.

(b) Phase de clustering.

FIGURE 6 – Discrétisation par ABBA. (Elsworth and Güttel, 2020)

2 Contribution

Après la revue des méthodes existantes d'apprentissage automatique d'AT, un nouvel algorithme, SYNTHESIS, combinant des caractéristiques de ces dernières a été développé. À l'instar des algorithmes de l'état de l'art, il utilise des traces horodatées de symboles. Le lien entre ST et traces horodatées sera présenté dans la section 3.2 lors des expérimentations.

2.1 Un nouvel algorithme d'apprentissage automatique d'AT

L'algorithme SYNTHESIS prend en entrée un set de séquences de symboles (ou événements) avec le délai se produisant entre chaque. Il retourne un PDRTA cohérent avec les données d'entrée. La stratégie globale est présentée dans l'algorithme 1. Les trois opérations, la création de l'APTA (inspirée par RTI+), la fusion d'états (inspirée par Timed k-Tail) et la division de transitions (inspirée par RTI+), sont respectivement détaillées dans les sections 2.2, 2.3 et 2.4. Les paramètres probabilistes de RTI+ ont également été conservés, non pas dans le but de les utiliser pour des calculs mais pour l'intérêt qu'ils peuvent avoir dans l'exploitation des automates résultants. La méthode se distingue par la combinaison et l'enchaînement de ces opérations. En premier lieu, l'automate est initialisé par la création d'un APTA. Une première phase consiste en la fusion des états de l'APTA semblant correspondre à un unique état du système à modéliser. La seconde phase consiste à utiliser les données temporelles pour réaliser des divisions de transition. À la suite de ces modifications, de nouvelles possibilités de fusion d'états sont recherchées. Durant cette seconde phase, les divisons de transitions sont favorisées et l'algorithme s'arrête lorsque plus aucune division de transition ou fusion d'états n'est possible.

Algorithme 1 SYNTHESIS : Apprentissage de PDRTA à partir de données positives

Requiert : un set de séquences positives $S = \{S_+\}$
Retourne : A est un PDRTA cohérent S
  $A =$ `apta`(S)
  tant que il existe des états au même k-futur **fait**
    `fusionne`
  fin tant que
  tant que le processus n'est pas terminé **fait**
    `divise` une transition divisable
    si aucune transition n'est divisable **alors**
      `fusionne` les états au même k-futur
      si aucuns états ne peuvent être fusionnés **alors**
        Le processus est terminé
      fin si
    fin si
  fin tant que

2.2 Construction de l'APTA

A l'instar de RTI+, la première étape de l'algorithme est la création d'un automate en forme d'arbre (voir algorithme 2) nommé APTA. Dans cet APTA, il existe exactement un chemin menant à chaque état. Tout d'abord, un état initial est créé. En prenant chaque séquence des données d'entrée, en commençant au niveau de l'état initial, on recherche s'il existe une transition de même symbole que celui observé dans la séquence. Le cas échéant, la transition est empruntée et le délai observé est ajouté à l'ensemble des valeurs observées pour la transition. Dans le cas contraire, la transition et son état de destination sont créés. L'état de destination de la transition correspond au nouvel état courant pour le symbole suivant dans la séquence.

Algorithme 2 Construction de l'APTA : `apta`

Requiert : un set de séquences positives $S = \{S_+\}$
Retourne : A est un automate en forme d'arbre cohérent avec S
  A est un automate avec un unique état q_0
  pour chaque séquence σ de S **fait**
    $q = q_0$
    pour chaque symbole a et sa valeur temporelle associées t dans σ **fait**
      si q a une transition sortante e avec a comme symbole **alors**
        si t n'est pas inclus dans la garde de e **alors** élargie e **fin si**
      sinon
        Créé une nouvelle transition e avec a comme symbole et $[t, t]$ comme garde
      fin si
      $q =$ l'état de destination de e
    fin pour
  fin pour

2.3 Fusion d'états

Afin de réduire la taille de l'automate en termes de nombre d'états, l'algorithme va chercher à fusionner les états semblant correspondre à un même état dans le système à

modéliser. Deux états de l'automates sont considérés comme étant un unique état dans le système s'ils offrent le même choix de symboles de transitions futures en partant de ceux-ci. Ces possibilités de symboles de transitions futures sont nommées le « k-futur » de l'état et le « k » se réfère au nombre de symbole de transitions à regarder au-delà de l'état. Par exemple, dans la figure 7, le k-futur de l'état « S0 » avec $k = 2$ est le set suivant : $\{"ab", "ba", "bc"\}$.

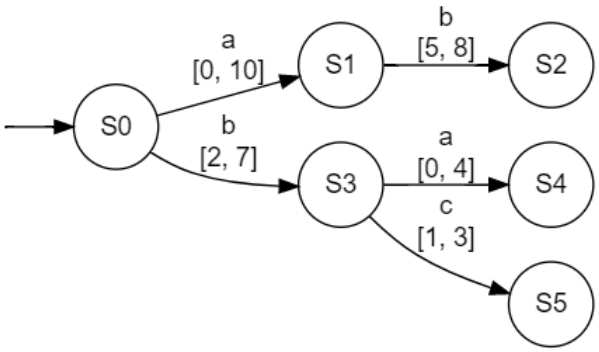


FIGURE 7 – Un APTA.

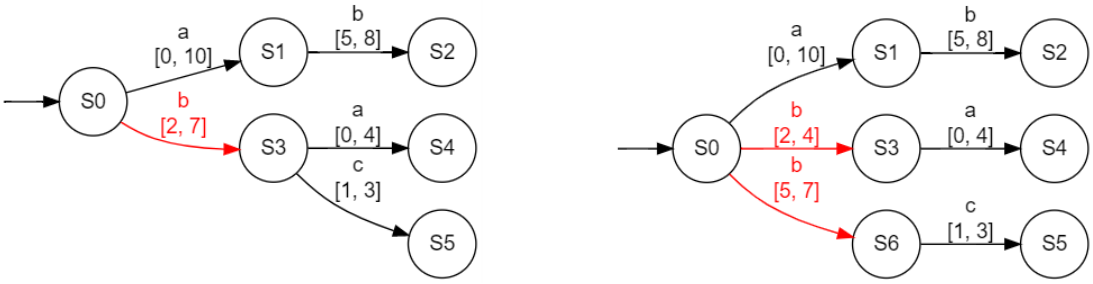
La valeur par défaut pour ce paramètre a été fixée à $k = 2$ mais reste modifiable. Les k-futurs de chaque état de l'automate sont calculés et les états ayant le même k-futur sont fusionnés. La fusion des deux états consiste en l'accumulation des transitions entrantes et sortantes des deux états sur le premier et en la suppression du second. Si deux transitions entrantes sont de même symbole et de même source, ces transitions sont elle-même fusionnées. La fusion des transitions engendre le cumul de l'ensemble de délais acceptés et donc l'élargissement de l'intervalle de valeur acceptées. Le paramètre probabiliste (occurrence d'emprunt en partant de l'état source) est aussi recalculé. Si le nouveau set de transition sortantes induit un choix non déterministe (même symbole lors de la première phase et même symbole ainsi que garde se chevauchant lors de la seconde phase lorsque les divisions de transition et les fusions d'états s'alternent), les transitions problématiques ainsi que leurs états de destinations sont aussi fusionnées, et ce jusqu'à la résolution de tout non-déterminisme induit dans l'automate. C'est le processus de déterminisation. L'algorithme 3 présente la fusion d'état et le processus de déterminisation.

Algorithme 3 Fusion d'états et process de déterminisation : `fusionne`

Requiert : Deux états q, q' de A, le nombre de transitions k à considérer
Retourne : A est un PDRTA cohérent avec S, déterministe et où q et q' sont fusionnés
  si q et q' ont le même set de séquences de k symboles possibles **alors**
    pour chaque transition sortante e de q' **fait**
      La nouvelle source de e est q
    fin pour
    pour chaque transition entrante e de q' **fait**
      La nouvelle destination de e est q
    fin pour
    tant que A a des transitions avec le même état source, le même symbole et respectivement q_n et q'_n comme états de destination **fait**
      `fusionne` q_n et q'_n
    fin tant que
  fin si

2.4 Division de transition

Une transition e_0 est dite « divisable » s'il s'agit de l'unique transition entrante de son état de destination, et si lorsque l'on considère la transition suivante visitée e_i, les intervalle de valeurs acceptées pour e_0 pour chaque transition e_i ne se chevauchent pas. Par exemple dans la figure 8, le set de séquences des données d'entrée empruntant la transition entre l'état « S0 » et « S3 » (en rouge) est le suivant : $\{$"$b:2\ a:0$","$b:4\ a:4$","$b:5\ c:1$","$b:7\ c:3$"$\}$. L'intervalle de valeurs temporelles acceptées associé à la transition en rouge si la transition empruntée en suivant est de symbole « c » est $[2, 4]$ et l'intervalle de valeurs acceptées si la transition empruntée en suivant est de symbole « a » est $[5, 7]$. Ces intervalles ne se chevauchant pas, la transition en rouge peut être divisée, menant à la création d'un nouvel état (« S6 »). L'algorithme 4 présente la division de transition.



(a) Automate avant la division.

(b) Automate après la division.

FIGURE 8 – Une division de transition.

Algorithme 4 Division de transition : `divise`

Requiert : Une transition e de A, unique transition a son état de destination q, qui a lui-même plusieurs transitions sortantes $f_1, ..., f_n$

Retourne : A est un PDRTA cohérent avec S, déterministe, sans gardes se chevauchant pour deux transitions avec le même état source et le même symbole

pour chaque séquence σ empruntant e **fait**
  Obtient la valeur temporelle associée à cet emprunt de e
  Obtient la transition f_i visitée après
fin pour
si les séquence empruntant f_i empruntent e dans un intervalle spécifique de valeurs temporelles g qui ne chevauche pas les intervalles spécifiques aux autres transitions pouvant être visitées après **alors**
  Créé un nouvel état q'
  Créé une nouvelle transition e' de même symbole et de même source que e, q' en tant qu'état de destination et une garde égale à g
  Change l'état source de f_i de q à q'
  Extrait g de la garde de q
fin si

SYNTHESIS a été codé en Python et plusieurs expérimentations ont été menées afin de le tester et de le comparer aux autres algorithmes de l'état de l'art (hormis GenProgTA car l'implémentation distribuée ne permet pas d'apprendre d'automate avec des nouveaux jeux de données). La section suivante présente certaines de ces expérimentations ainsi que les résultats obtenus les plus importants.

3 Expérimentation

Dans un premier temps, l'algorithme est testé et comparé aux autres sur un jeu de séquences d'événements temporisées obtenues à partir d'un automate modèle. Dans un second temps, des ST sont générées à partir d'un système fictif, elles sont discrétisées par les différentes méthodes présentées dans l'état de l'art puis les jeux de données obtenus sont utilisés sur les méthodes d'apprentissage automatique d'AT afin de comparer l'ensemble des approches.

3.1 Nouvel algorithme sur les traces d'un automate modèle

3.1.1 Jeu de données

Afin de tester la capacité de l'algorithme à identifier la structure d'un AT, nous avons généré un set de séquences (traces de symboles et de délais) à partir d'un automate modèle. L'automate modèle est un DRTA (Figure 9) avec cinq états et un cas de déterminisme temporel : deux transitions de même symbole avec le même état source (« c1 ») mais des gardes ne se chevauchant pas. 2000 séquences temporisées ont été générées, partant de l'état initial (« c0 ») et s'arrêtant aléatoirement sur un des états de l'automate. La transition empruntée au sortir d'un état et le délai associé ont été choisis de manière aléatoire et uniforme entre tous les cas possibles.

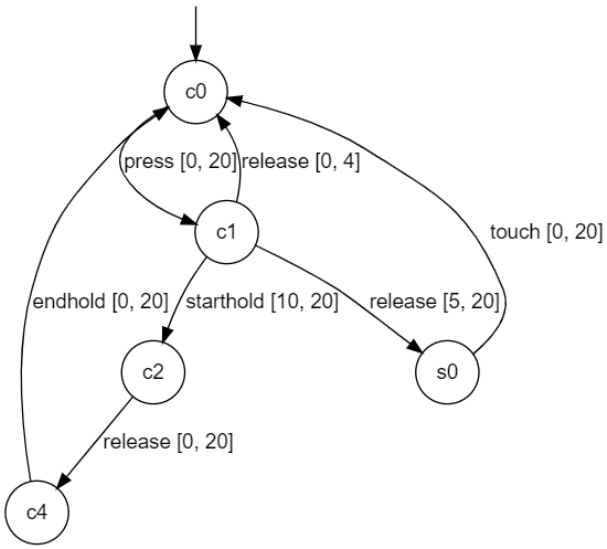


FIGURE 9 – DRTA modèle.

3.1.2 Résultats

SYNTHESIS a correctement identifié la structure de l'automate et ses contraintes temporelles en neuf secondes, en renvoyant un automate identique à celui de la figure 9. Le paramètre k avait été fixé à se valeur par défaut soit $k = 2$.

Les autres algorithmes (RTI+ et Time k-Tail) n'ont pas retrouvé cette même structure et ces mêmes contraintes temporelles. Quelques caractéristiques de ces automates sont comparées dans le tableau 1.

Algorithme	Nombre d'états	Nombre de transitions	Déterministe	Cohérent avec les données
SYNTHESIS	5	7	Oui	Oui
Time k-Tail	6	11	Non	Oui
RTI+	6	8	Oui	Oui

Tableau 1 – Caractéristiques des automates résultants.

3.2 Comparaison des algorithmes d'apprentissage d'AT à partir de séries temporelles

Afin de comparer les algorithmes de la littérature avec le nouvel algorithme proposé, et de vérifier la possibilité de modéliser un système à partir de ST, nous avons imaginé un système et généré des ST qui aurait pu en être issues. Plusieurs méthodes de symbolisation de ST présentées dans l'état de l'art ont été utilisées pour obtenir les séquences de symboles et de délais. Chaque jeu de séquences correspondant à chaque méthode de discrétisation a été utilisé comme jeu de données d'entrée pour chaque algorithme testé. Cependant, nous ne présenterons ici que les AT obtenus par chaque algorithme avec la méthode de discrétisation sélectionnée au terme de l'expérimentation.

3.2.1 Système original

Considérons une ligne de conditionnement utilisée pour mettre des jus en bouteille. Le système d'injection de produit est équipé d'une sonde mesurant et enregistrant le pH de manière continue afin d'assurer la qualité du produit. Cette ligne est aussi équipée d'un système de Nettoyage En Place (NEP), un système automatique de nettoyage interne de la ligne avec une solution alcaline et une solution acide. Le plan d'utilisation de la ligne est présenté sur la figure 10.

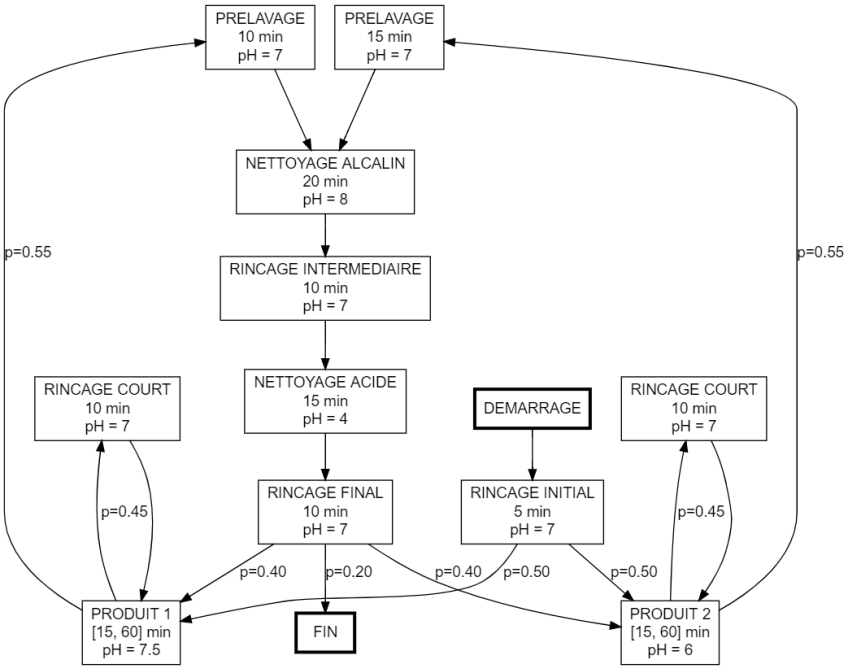


FIGURE 10 – Plan de production de la ligne d'embouteillage.

Au démarrage de la ligne, un court rinçage est effectué avant le début de l'embouteillage. La durée d'embouteillage est variable. La ligne est rincée au minimum toutes les heures au cours de la production. A tout changement de produit à embouteiller, un nettoyage par NEP est réalisé. Entre l'injection de l'alcalin et de l'acide, la ligne doit être rincée pour assurer l'efficacité du nettoyage. Enfin, la ligne doit aussi être nettoyée par NEP avant son arrêt.

Chaque flux passant par le système d'injection a un pH spécifique :

— Produit 1 : $pH = 7.5$,
— Produit 2 : $pH = 6$,
— Eau : $pH = 7$,
— Solution alcaline : $pH = 8$,
— Solution acide : $pH = 4$.

Nous considérons ici un système parfait sans transitions entre les valeurs de pH (dues au reste de produit dans la ligne).

3.2.2 Génération de séries temporelles

En se basant sur le plan de production de la figure 10, nous avons généré 200 ST de l'évolution du pH au cours du temps. Nous avons ajouté un bruit gaussien de paramètres μ= 0 et σ^2= 0.05 pour simuler l'imprécision de la sonde. La figure 11 présente une de ces ST.

70% du jeu de données obtenu après symbolisation sera utilisé pour apprendre un AT avec SYNTHESIS, RTI+ et Timed k-Tail. Afin de valider les automates résultants, les 30% restant seront utilisés pour calculer la sensibilité des modèles. Nous avons par ailleurs généré 60 ST ne correspondant pas au système : 30 ST avec des durées non conformes au plan de production et 30 séries avec des enchaînements de valeur de pH impossibles à observer conformément au plan. Ces séries seront utilisées pour calculer la spécificité des modèles.

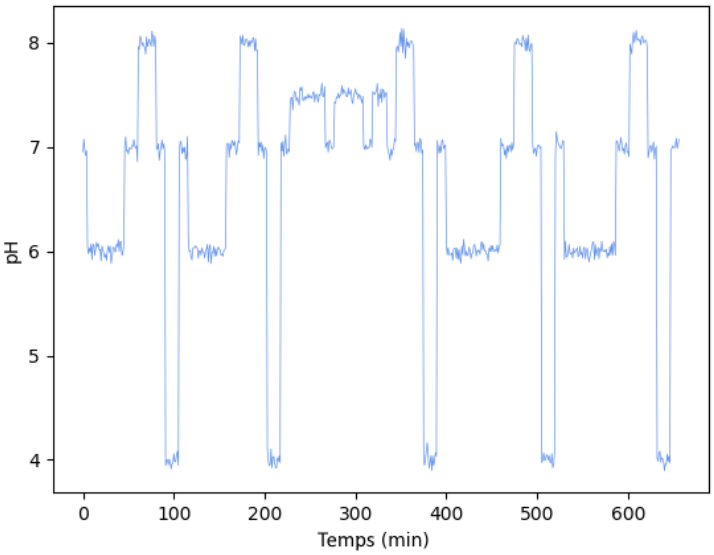


FIGURE 11 – Valeur du pH au cours du temps.

3.2.3 Discrétisation des séries temporelles

Le processus d'apprentissage automatique d'AT requiert des données discrètes sous forme de séquence d'événements et de délais. Par conséquent, les ST doivent être prétraitées avec une étape de discrétisation sans que l'on connaisse les symboles à priori. Plu-

sieurs méthodes présentées dans l'état de l'art (section 1.2) ont été testées et les séquences obtenues ont été utilisées comme données d'entrée pour chaque algorithme d'apprentissage d'AT afin de voir quelle méthode aboutissait aux automates les plus informatifs et interprétables.

SAX est la méthode de discrétisation de ST la plus connue. La figure 12 présente la ST de la figure 11 avec chaque point de la série associé à son cluster après utilisation de SAX. Cette méthode à mi-chemin entre les méthodes statiques et dynamiques aboutit a un partionnement ne semblant pas pertinent. En effet, pour un même produit, les points vont être classés dans des clusters différents à cause des variations du pH dues au bruit, et ce malgré la PAA.

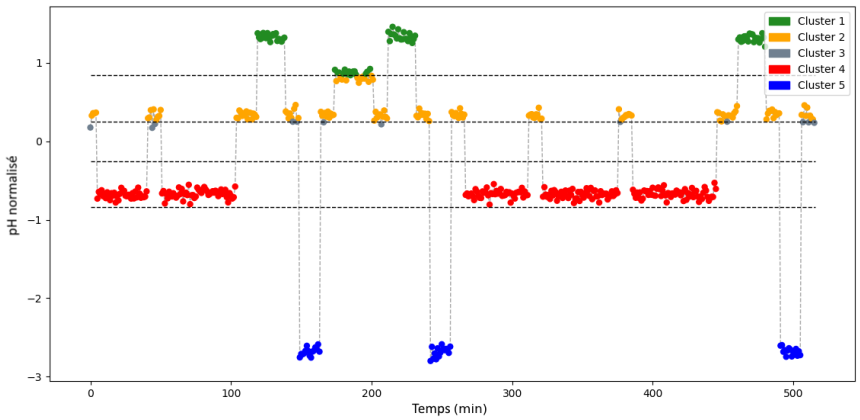


FIGURE 12 – Clusters obtenus avec SAX.

ABBA est conçu pour identifier des motifs représentant une évolution de valeur par sa durée et sa valeur d'incrémentation. Il peut être intéressant d'obtenir une telle symbolisation pour nos données, en identifiant les phases de changement de flux et les phases de continuité. Après la phase de réduction de dimension où différent segments ont été identifiés, ABBA a partitionné les segments en huit clusters comme le montre la figure 13. Cette partition ne paraît pas optimale au vu de la nature des données puisque chaque « palier » de pH peut être divisé en plusieurs clusters et que cette division n'est pas identique d'un palier à un autre pour une même valeur de pH. De plus, cet algorithme ne traite qu'une ST à la fois, il n'est pas possible d'apprendre et d'appliquer le modèle sur l'ensemble de nos données. Par conséquent, il n'est pas possible de générer un jeu de données pour l'apprentissage d'AT avec ABBA.

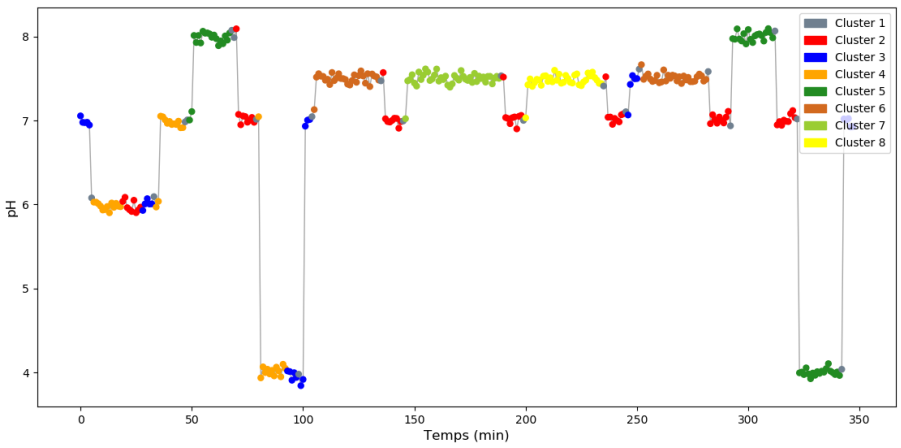


FIGURE 13 – Clusters obtenus avec ABBA.

Le clustering par les k-Means, méthode de symbolisation statique, permet d'obtenir des symboles interprétables (voir figure 14), chaque symbole correspondant à la valeur de pH de chaque flux. Le nombre de clusters a été obtenu en calculant le DBI pour un nombre de cluster allant de 2 à 15. Ce clustering a permis une suppression du bruit mais a pu être efficace que grâce au non-chevauchement des intervalles de valeur de pH pour un produit malgré l'ajout de bruit. L'élimination du bruit sans erreur a pu être confirmée par la comparaison des résultats avec les ST non discrétisées.

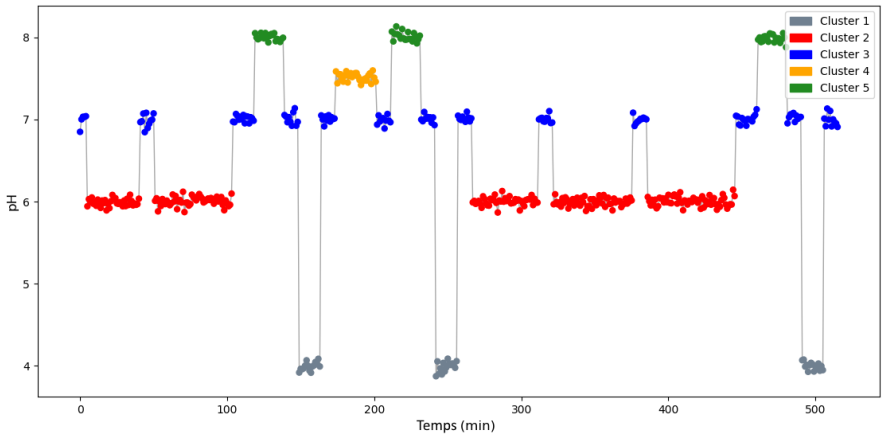


FIGURE 14 – Clusters obtenus avec les k-Means.

La modélisation d'un MMC a permis l'obtention des mêmes symboles (états cachés) directement interprétables que le clustering par les k-Means (voir figure 15). Les jeux de données obtenu sont donc identiques.

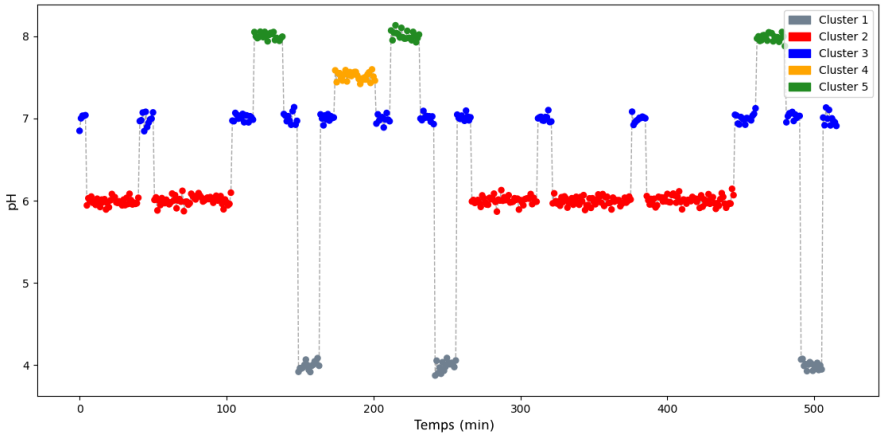


FIGURE 15 – Clusters obtenus avec la méthode MMC.

Les ST discrétisées par SAX et le MMC (ou k-Means) ont finalement été transformées en séquence de symboles associés au délai au bout duquel le changement de symbole a eu lieu. Par exemple, « *aaabccccccaa* » donne « $a:0\ b:3\ c:1\ a:2$ » après transformation, ce qui signifie que le symbole « a » apparaît dès le début de la série, qu'il change pour « b » au bout de trois unités de temps et ainsi de suite.

3.2.4 Résultats

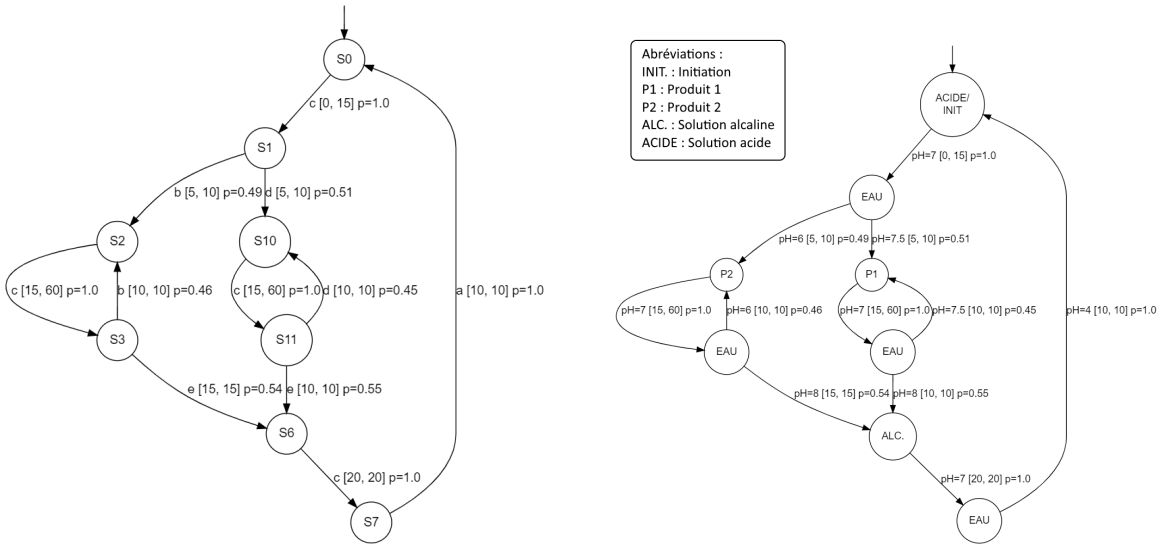
Le jeu de données ayant donné les automates les plus interprétables et généralement les meilleurs scores de validation (sensibilité et spécificité) est celui généré par la méthode des

k-Means et la méthode des MMC. Les scores de validation de l'ensemble des combinaisons sont présentés dans le tableau 2 mais seuls les automates issus du jeu de données MMC/k-Means sont présentés dans cette section.

Algorithme d'apprentissage d'AT	**Méthode de discrétisation**	**Sensibilité**	**Spécificité**
SYNTHESIS	SAX	0.12	0.98
	MMC/k-Means	1.00	1.00
Timed k-Tail	SAX	?	?
	MMC/k-Means	?	?
RTI+	SAX	0.57	0.05
	MMC/k-Means	0.50	1.00

Tableau 2 – Scores de validation des modèles appris. Les automates obtenus par Time k-Tail étant non déterministes, il est impossible de calculer ces scores.

SYNTHESIS a appris l'AT présenté dans la figure 16a en sept secondes. Le paramètre k a été fixé sur sa valeur par défaut ($k = 2$). Cet automate accepte toutes les séquences du set de validation et rejette toutes les séquences du set de séquences erronées. Chaque symbole correspondant à une valeur de pH et les transitions entrant dans chaque état étant de même symbole, il est possible d'interpréter les états de l'automate tel que sur la figure 16b. Les états du plan de production sont correctement identifiés et les contraintes temporelles correspondent bien à la durée possible de chaque phase, excepté pour la transition partant de l'état « S0 » et allant à l'état « S1 ». Ceci est dû à la fusion de l'état initial avec l'état correspondant à la phase de nettoyage acide. Cette fusion a été réalisée car ces phases sont suivies par les mêmes phases à savoir un rinçage puis une phase d'embouteillage, ce n'est donc pas une erreur de sémantique.



(a) PDRTA résultant. (b) Interprétation des symboles et des états.

FIGURE 16 – Automate appris par SYNTHESIS avec MMC/k-Means.

Timed k-Tail a appris son automate en trois secondes. Le formalisme utilisé par Timed k-Tail étant conçu pour apprendre des systèmes avec des opérations imbriquées, l'automate obtenu considère pour chaque occurrence de symbole deux transitions (début et

fin d'occurrence) et une horloge différente réinitialisée sur la première transition et avec une contrainte temporelle sur la seconde. N'étant pas dans un cas où les opérations sont imbriquées, on peut simplifier l'automate obtenu tel que présenté sur la figure 17. Cet automate n'est pas déterministe, ce qui empêche en empêche sa validation. Il est trop général, non interprétable mais permet de mettre en avant que chaque phase autre que rinçage en est précédée par un.

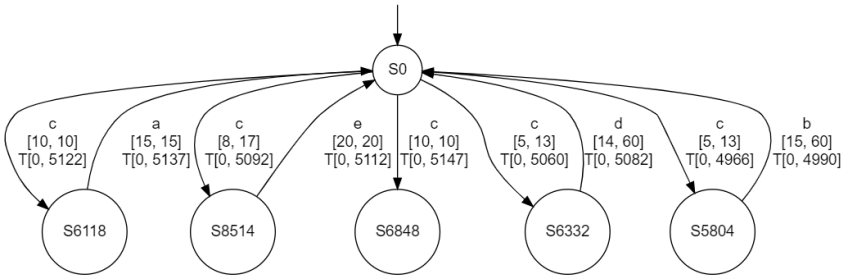


FIGURE 17 – Automate simplifié obtenu par Timed k-Tail avec MMC/k-Means.

En deux secondes, RTI+ a appris un automate dont la version interprétée (de la même manière que celui obtenu avec SYNTHESIS) est présentée dans la figure 18. Cet automate accepte toutes les séquences du set de validation. Ses états sont interprétables ce qui permet de dire que la structure est bonne, mais il y a un manque de fusion d'états ce qui occasionne une duplication des états du processus de NEP. Le second problème réside dans les contraintes temporelles. RTI+ utilise la valeur temporelle minimum et le maximum dans tout le jeu de données comme limite inférieure et supérieure des gardes des transitions. Il n'y a donc une perte élevée de l'information temporelle. De ce fait, l'automate obtenu par RTI+ refuse bien les séquences erronées où des transitions inexistantes avaient été ajoutées. Cependant, étant donné le problème soulevé sur ses gardes, il accepte à tort les séquences négatives pour lesquelles les durées des phases avaient été modifiées.

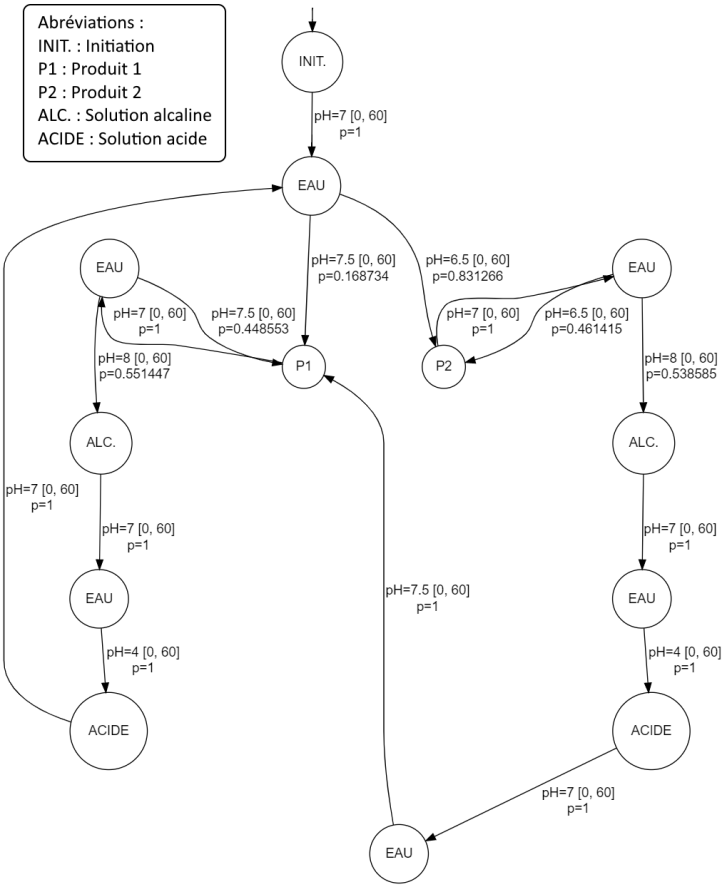


FIGURE 18 – Automate interprété obtenu par RTI+ avec MMC/k-Means.

4 Discussion

4.1 SYNTHESIS

4.1.1 Vérification de l'algorithme

La validation de l'algorithme développé est à l'heure actuelle uniquement empirique. Afin de prouver la justesse de ce dernier, il sera nécessaire de réaliser une vérification formelle avec notamment une étude théorique de la convergence.

4.1.2 Processus de déterminisation

Le processus de déterminisation proposé dans RTI+ et réutilisé dans SYNTHESIS permet de résoudre les situations de non-déterminisme induites par la fusion d'états au même k-future. Les états fusionnés au cours de ce processus n'ont quant à eux pas obligatoirement le même futur ce qui peut nous amener à nous questionner sur le bien-fondé de la démarche. Une étude de la défaisabilité de ces fusions par division de transition doit être réalisée afin de vérifier que le processus de déterminisation n'occasionne pas d'erreurs permanentes.

4.1.3 Paramètres

La valeur par défaut pour le paramètre k a été fixée à $k = 2$ par calquage sur Time k-Tail. Aucune étude n'a été menée pour déterminer une valeur qui donnerait le meilleur résultat dans une majorité de cas. Une valeur élevée conduirait à des automates avec un

plus grand nombre d'états mais une valeur plus faible pourrait mener à un modèle trop général. Le choix de la valeur du paramètre k renvoie donc au dilemme biais-variance et la « meilleure » valeur semble être propre à chaque cas.

4.1.4 Temps d'exécution

Le temps nécessaire à l'obtention du résultat avec SYNTHESIS est supérieur à celui nécessaire aux deux autres algorithmes testés pour un même jeu de données. RTI+ paraît pourtant plus complexe puisqu'il réalise des calculs de vraisemblance ainsi que des tests de significativité. Ce point peut être amélioré en changeant de langage de codage, par exemple pour C++ qui est utilisé pour RTI+. Cependant, le temps d'exécution actuel semble raisonnable et adapté à un usage en industrie.

4.2 Apprentissage d'AT à partir de séries temporelles : choix du système

Dans le système considéré dans la section 3.2, la variable choisie (valeur du pH) pour obtenir les ST induit une segmentation dans l'espace des valeurs, créant ainsi un symbole par phase de production. De plus, les transitions entre palier de valeur ne sont pas considérées alors que le pH devrait évoluer progressivement du fait de l'effet de dilution du reste de chaque produit dans le produit suivant. C'est donc un cas simpliste où la discrétisation n'implique pas nécessairement de méthode spécifique à la prise en compte de la dimension temporelle.

4.3 Méthode de validation des automates

4.3.1 Automates obtenus à partir de séquences d'un automate modèle

Dans la section 3.1, la validation de l'AT résultant de l'apprentissage à partir de séquences obtenues sur un automate modèle est évidente car le résultat et le modèle sont strictement identiques (même structure, mêmes gardes et étiquettes sur les transitions). Dans le cas où l'automate obtenu différerait, il serait intéressant d'avoir un moyen de mesurer la distance entre le résultat et le modèle. Un premier calcul de distance pouvant être effectué est la distance d'édition entre deux traces d'AT (Chatterjee et al., 2014). Elle consiste en le coût minimum d'une séquence d'opérations (insertion, suppression ou substitution de symbole ou de valeur d'horloge) à réaliser pour transformer une trace en une autre. Le calcul de la distance d'édition entre deux automates pourrait être réalisé en comparant toutes les traces possibles d'un premier AT avec toutes les traces possibles du second AT, cependant ce problème est indécidable mais une approximation de cette distance est proposée (Chatterjee et al., 2014). Cependant, le set de traces acceptées par notre automate modèle est infini, comme dans tous les cas où il n'y a pas de passage obligatoire par un état final. Il pourrait être intéressant de rechercher une adaptation à la distance d'édition par approximation.

Une autre manière de calculer la distance entre deux AT est de calculer à quel point le set de traces du premier automate est inclus dans le set de traces du suivant. A ce jour, il est possible de déterminer si cette distance est finie ou infinie (Rosenmann, 2019) mais pas d'obtenir une quantification de cette distance dans le cas où elle serait finie. La remarque précédente sur le set de traces des automates étant souvent infini s'applique de nouveau pour cette distance.

4.3.2 Automates obtenus à partir de séries temporelles

La validation des automates obtenus avec les différents jeux de données et les différents algorithmes reste contestable. En effet, si le calcul de la sensibilité et de la spécificité de chaque modèle permet d'obtenir des valeurs chiffrées sur lesquelles s'appuyer, l'interprétabilité des automates est jugée de manière relative. Or, dans le cadre de systèmes réels dont on ne connaît pas le fonctionnement, il est rarement possible d'avoir à disposition des séquences erronées.

Conclusion

L'objectif de ce travail était la recherche d'une méthode d'apprentissage automatique d'AT à partir de séries temporelles de sorte à réduire le besoin d'intervention humaine dans le processus de construction de ces automates et d'exploiter les données de capteurs ou de logs récupérés au cours des process.

Nous avons tout d'abord réalisé un état de l'art des méthodes existantes d'apprentissage automatique d'AT. Ce problème étant NP-difficile, les auteurs ont choisi de se concentrer sur l'apprentissage de sous-classes d'AT (RTI+ et Time k-Tail) ou de passer par la programmation génétique (GenProgTA). Tandis que RTI+ passe par l'utilisation d'une heuristique pour décider des opérations a effectuer, Time k-Tail est une approche algorithmique sans vérification formelle. Les méthodes étudiées dans cet état de l'art ont été testées et évaluées, et la contribution a été de proposer un nouvel algorithme permettant de pallier certaines erreurs identifiées. Nous avons ensuite vérifié que cet algorithme était capable de réidentifier un automate modèle à partir d'un set de traces en étant issu.

Parallèlement, nous avons réalisé un état de l'art des méthodes de discrétisation des ST, l'AT étant une représentation à événements discrets. Plusieurs méthodes ont été identifiées, certaines statiques à savoir qu'elles ne prenaient pas en compte la dimension temporelles et d'autres dynamiques qui considéraient l'évolution de la variable et non uniquement la répartition des valeurs qu'elle pouvait prendre.

Nous avons finalement imaginé un système et généré des ST qui auraient pu en être issues. Les méthodes de discrétisation de ST ont été appliquées à ces données et les différents jeu de données obtenus ont été mis en entrée des algorithmes d'apprentissage d'AT de l'état de l'art ainsi que du nouveau. La validation des automates a été réalisée de manière subjective avec une tentative d'interpréter les automates obtenus, et de manière objective avec le calcul de la spécificité et de la sensibilité de chaque modèle. Le nouvel algorithme avec les données discrétisées par la méthode des k-Means ou du MMC a donné le meilleur résultat au niveau de l'interprétabilité et des scores calculés, dans le cas de notre système fictif.

L'algorithme semble être capable d'apprendre un automate cohérent avec les données et pourrait trouver des applications en industrie, par exemple en aide au diagnostic de panne ou en anticipation pour prendre des décisions. La validation formelle de l'algorithme reste à réaliser et l'expérimentation devrait être menée sur d'autres données avec les différentes méthodes de discrétisation de ST. Par la suite, il pourrait être utile de tenter de se rapprocher de la définition initiale d'AT, en commençant par intégrer l'identification de plusieurs horloges puisque l'algorithme apprend actuellement un automate où la notion d'horloge est réduite au délai écoulé entre deux événements. Il pourrait aussi être intéressant de se pencher sur l'apprentissage de réseaux d'AT, ces derniers étant très utilisés pour modéliser des systèmes complexes tel que des processus biologiques de régulation (Ben Abdallah et al., 2016).

Références

Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2) :183–235, April 1994. ISSN 0304-3975. doi : 10.1016/0304-3975(94)90010-8. URL http://www.sciencedirect.com/science/article/pii/0304397594900108.

Emna Ben Abdallah, Tony Ribeiro, Morgan Magnin, Olivier Roux, and Katsumi Inoue. Inference of Delayed Biological Regulatory Networks from Time Series Data. In Ezio Bartocci, Pietro Lio, and Nicola Paoletti, editors, *Computational Methods in Systems Biology*, volume 9859, pages 30–48. Springer International Publishing, Cham, 2016. ISBN 978-3-319-45176-3 978-3-319-45177-0. doi : 10.1007/978-3-319-45177-0_3. URL http://link.springer.com/10.1007/978-3-319-45177-0_3. Series Title : Lecture Notes in Computer Science.

M. Biba, F. Esposito, S. Ferilli, N. Mauro, and T. Basile. Unsupervised Discretization Using Kernel Density Estimation. In *IJCAI*, 2007.

A. W. Biermann and J. A. Feldman. On the Synthesis of Finite-State Machines from Samples of Their Behavior. *IEEE Transactions on Computers*, C-21(6) :592–597, June 1972. ISSN 0018-9340. doi : 10.1109/TC.1972.5009015. URL http://ieeexplore.ieee.org/document/5009015/.

Krishnendu Chatterjee, Rasmus Ibsen-Jensen, and Rupak Majumdar. Edit distance for timed automata. In *Proceedings of the 17th international conference on Hybrid systems : computation and control - HSCC '14*, pages 303–312, Berlin, Germany, 2014. ACM Press. ISBN 978-1-4503-2732-9. doi : 10.1145/2562059.2562141. URL http://dl.acm.org/citation.cfm?doid=2562059.2562141.

Steven Elsworth and Stefan Güttel. ABBA : adaptive Brownian bridge-based symbolic aggregation of time series. *Data Mining and Knowledge Discovery*, June 2020. ISSN 1384-5810, 1573-756X. doi : 10.1007/s10618-020-00689-6. URL http://link.springer.com/10.1007/s10618-020-00689-6.

Allan Grønlund, Kasper Green Larsen, Alexander Mathiasen, Jesper Sindahl Nielsen, Stefan Schneider, and Mingzhou Song. Fast Exact k-Means, k-Medians and Bregman Divergence Clustering in 1D. *arXiv :1701.07204 [cs]*, April 2018. URL http://arxiv.org/abs/1701.07204. arXiv : 1701.07204.

Mirosław Kurkowski, Olga Siedlecka-Lamch, and Paweł Dudek. Using Backward Induction Techniques in (Timed) Security Protocols Verification. In Khalid Saeed, Rituparna Chaki, Agostino Cortesi, and Sławomir Wierzchoń, editors, *Computer Information Systems and Industrial Management*, volume 8104, pages 265–276. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-40924-0 978-3-642-40925-7. doi : 10.1007/978-3-642-40925-7_25. URL http://link.springer.com/10.1007/978-3-642-40925-7_25. Series Title : Lecture Notes in Computer Science.

Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In Jaime G. Carbonell, Jörg Siekmann, G. Goos, J. Hartmanis, J. van Leeuwen, Vasant Honavar, and Giora Slutzki, editors, *Grammatical Inference*, volume 1433, pages 1–12. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-64776-8 978-3-540-68707-8. doi : 10.1007/BFb0054059. URL http://link.springer.com/10.1007/BFb0054059. Series Title : Lecture Notes in Computer Science.

Christine Largouët, Marie-Odile Cordier, Yves-Marie Bozec, Yulong Zhao, and Guy Fontenelle. Use of timed automata and model-checking to explore scenarios on ecosystem models. *Environmental Modelling & Software*, 30 :123–138, April 2012. ISSN 13648152. doi : 10.1016/j.envsoft.2011.08.005. URL https://linkinghub.elsevier.com/retrieve/pii/S1364815211001885.

Richard J. Larsen and Morris L. Marx. *An introduction to mathematical statistics and its applications*. Prentice Hall, Boston, 5th ed edition, 1986. ISBN 978-0-321-69394-5. OCLC : ocn502674159.

Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery - DMKD '03*, page 2, San Diego, California, 2003. ACM Press. doi : 10.1145/882082.882086. URL http://portal.acm.org/citation.cfm?doid=882082.882086.

Fabian Mörchen and Alfred Ultsch. Optimizing time series discretization for knowledge discovery. In *Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining - KDD '05*, page 660, Chicago, Illinois, USA, 2005. ACM Press. ISBN 978-1-59593-135-1. doi : 10.1145/1081870.1081953. URL http://portal.acm.org/citation.cfm?doid=1081870.1081953.

Fabrizio Pastore, Daniela Micucci, and Leonardo Mariani. Timed k-Tail : Automatic Inference of Timed Automata. *arXiv :1705.08399 [cs]*, May 2017. URL http://arxiv.org/abs/1705.08399. arXiv : 1705.08399.

Amnon Rosenmann. On the Distance Between Timed Automata. In Étienne André and Mariëlle Stoelinga, editors, *Formal Modeling and Analysis of Timed Systems*, volume 11750, pages 199–215. Springer International Publishing, Cham, 2019. ISBN 978-3-030-29661-2 978-3-030-29662-9. doi : 10.1007/978-3-030-29662-9_12. URL http://link.springer.com/10.1007/978-3-030-29662-9_12. Series Title : Lecture Notes in Computer Science.

Anita Sant'Anna and Nicholas Wickstrom. Symbolization of time-series : An evaluation of SAX, Persist, and ACA. In *2011 4th International Congress on Image and Signal Processing*, pages 2223–2228, Shanghai, October 2011. IEEE. ISBN 978-1-4244-9304-3 978-1-4244-9306-7. doi : 10.1109/CISP.2011.6100559. URL http://ieeexplore.ieee.org/document/6100559/.

Martin Tappler, Bernhard Aichernig, Kim Larsen, and Florian Lorber. *Learning Timed Automata via Genetic Programming*. August 2018.

Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. A Likelihood-Ratio Test for Identifying Probabilistic Deterministic Real-Time Automata from Positive Data. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, José M. Sempere, and Pedro García, editors, *Grammatical Inference : Theoretical Results and Applications*, volume 6339, pages 203–216. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-15487-4 978-3-642-15488-1. doi : 10.1007/978-3-642-15488-1_17. URL http://link.springer.com/10.1007/978-3-642-15488-1_17. Series Title : Lecture Notes in Computer Science.

Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. Efficiently identifying deterministic real-time automata from labeled data. *Machine Learning*, 86(3) :295–333, March 2012. ISSN 1573-0565. doi : 10.1007/s10994-011-5265-4. URL `https://doi.org/10.1007/s10994-011-5265-4`.

Willian Zalewski, Fabiano Silva, Huei Diana Lee, Andre Gustavo Maletzke, and Feng Chung Wu. Time Series Discretization Based on the Approximation of the Local Slope Information. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Juan Pavón, Néstor D. Duque-Méndez, and Rubén Fuentes-Fernández, editors, *Advances in Artificial Intelligence – IBERAMIA 2012*, volume 7637, pages 91–100. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-34653-8 978-3-642-34654-5. doi : 10.1007/978-3-642-34654-5_10. URL `http://link.springer.com/10.1007/978-3-642-34654-5_10`. Series Title : Lecture Notes in Computer Science.

l'institut Agro agriculture • alimentation • environnement	Diplôme : Ingénieur Agroalimentaire Spécialité : Science des données Spécialisation / option : Enseignant référent : Mathieu Emily

Auteur(s) : Lénaïg Cornanguer Date de naissance* : 22/10/1996	Organisme d'accueil : INRIA Adresse : Campus de Beaulieu, 263 Avenue Général Leclerc, 35042 Rennes
Nb pages : 20 Annexe(s) : 0	Maître de stage : Christine Largouët et Laurence Roze
Année de soutenance : 2020	

Titre français : Apprentissage automatique d'automates temporisés à partir de séries temporelles

Titre anglais : Time automata learning from time series

Résumé (1600 caractères maximum) : L'Automate Temporisé (AT) est une méthode formelle utilisée pour modéliser des systèmes dynamiques tels que des écosystèmes ou des protocoles de sécurité informatique. De nombreux outils ont été développés pour les AT, permettant de vérifier des propriétés de système avec le model checking, ou de générer des scénarios de test avec la synthèse de contrôleur. Ces modèles sont actuellement construits manuellement à partir des connaissances que les experts ont du système. Le travail présenté dans ce mémoire concerne l'utilisation de données issues de capteurs, des séries temporelles (ST), liés à un système pour la modélisation de ce dernier sous forme d'AT. La problématique est alors la recherche d'une méthode d'apprentissage automatique d'AT à partir de ST. Pour ce faire, l'état de l'art des algorithmes d'apprentissage automatique d'AT est effectué, ainsi que celui des méthodes de discrétisation des ST car l'AT repose sur des événements discrets. Les algorithmes sont comparés et un nouvel algorithme, SYNTHESIS, est développé. Différentes méthodes de discrétisation de l'état de l'art sont aussi comparées sur un jeu de données synthétique. Les jeux de données issus des méthodes de discrétisation sont utilisés pour apprendre des AT avec les différents algorithmes. C'est la discrétisation par un Modèle de Markov Caché ou par les k-Means, associée à l'apprentissage par SYNTHESIS, qui donne les meilleurs résultats. Les principales perspectives de ce travail sont la vérification formelle de SYNTHESIS et la recherche d'un outil de validation des AT appris.

Abstract (1600 caractères maximum) : The Time Automata (TA) is a formal method used to model dynamic systems such as ecosystems, electronic circuits, or security protocols. Numerous tools have been developed for the TAs, allowing to verify system properties expressed in mathematical logic with model checking or to generate test scenarios with controller synthesis. These models are currently built manually based on the experts' knowledge of the system. The work presented in this thesis concerns the use of data from sensors linked to a system for the modeling of the latter in the form of AT. This would make it possible to model systems which we do not know how they work. These sensor data often take the form of Time Series (TS). The problem is then the search for a method of automatic learning of TA from ST. With this aim in mind, the state of the art of TA machine learning algorithms is performed, as well as the state of the art of ST discretization methods because AT is based on discrete events. The algorithms are compared and a new algorithm, SYNTHESIS, is developed by taking the strengths of each one and trying to overcome their weaknesses. Different discretization methods of the state of the art are also compared on a synthetic dataset. The data sets from the discretization methods are used to learn TAs with the different algorithms including SYNTHESIS. The best results are obtain using a Hidden Markov Model or the k-Means to discretize the data, and SYNTHESIS to learn the TA. The main perspectives of this work are the formal verification of SYNTHESIS and the search for a validation tool for the learned TAs.

Mots-clés : Automate Temporisé, séries temporelles, Machine Learning, discrétisation

Key Words: Time Automata, time series, Machine Learning, discretization

* *Elément qui permet d'enregistrer les notices auteurs dans le catalogue des bibliothèques universitaires*